Weak Affine Light Typing is complete with respect to Safe Recursion on Notation

Luca Roversi [*][†]

Abstract

Weak affine light typing (WALT) assigns light affine linear formulae as types to a subset of λ-terms of System F. WALT is poly-time sound: if a λ-term M has type in WALT, M can be evaluated with a polynomial cost in the dimension of the derivation that gives it a type. The evaluation proceeds under any strategy of a rewriting relation which is a mix of both call-by-name and call-by-value β-reductions. WALT *weakens*, namely *generalizes*, the notion of "*stratification of deductions*", common to some Light Systems — those logical systems, derived from Linear logic, to characterize the set of Polynomial functions — . A weaker stratification allows to define a compositional *embedding* of Safe recursion on notation (SRN) into WALT. It turns out that the expressivity of WALT is strictly stronger than the one of the known Light Systems. The embedding passes through the representation of a subsystem of SRN. It is obtained by restricting the composition scheme of SRN to one that can only use its safe variables *linearly*. On one side, this suggests that SRN, in fact, can be redefined in terms of more primitive constructs. On the other, the *embedding* of SRN into WALT enjoys the two following remarkable aspects. Every datatype, required by the embedding, is represented from scratch, showing the strong structural proof-theoretical roots of WALT. Moreover, the embedding highlights a stratification structure of the normal and safe arguments, normally hidden inside the world of SRN-normal/safe variables: the less an argument is "polyomially impredicative", the deeper, in a formal, proof-theoretical sense, it is represented inside WALT. Finally, since WALT is SRN-complete it is also polynomial-time complete since SRN is.

[*]Dipartimento di Informatica, Università di Torino, C.so Svizzera n.ro 185 — 10149, TORINO — ITALY.
[†]*e-mail*:`roversi@di.unito.it`. *home page*:`http://www.di.unito.it/~rover`.

Contents

1 Introduction

Implicit computational complexity (ICC) explores machine-independent characterizations of complexity classes without any explicit reference to resource usage bounds, which, instead, result from restricting suitable computational structures. Contributions to ICC can have their roots in the recursion theory [Cob65, BC92, LM94, Lei95, Lei99, LM], in the structural proof-theory and linear logic [Gir98, Laf04], in the rewriting systems or functional programming [Hue80, Der82, Jon99, Lei93, Lei94], in the type systems [Hof97, Hof99a, Hof99b, Hof00, BNS00, BS01]. . . .

One specific goal of ICC is to make evident that the known complexity classes are concepts with an intrinsic mathematical nature. A way of achieving the goal is to formally relate the known ICC characterizations.

Here, we accomplish the goal relatively to two ICC systems that characterize the class FP of *Polynomial functions*. Specifically, we formally relate the basic concept of "predicative recursion" of Safe recursion on notation (SRN) [BC92], and the notion of "stratification of the derivations", basic for Light linear logic (LLL) [Gir98] and for various proof/type theoretical systems, derived from it. The strategy to formalize the relation is to embed, inductively, SRN into Weak affine light typing (WALT) [Rov07], a type system for pure λ-terms that strictly generalizes Intuitionistic light affine logic (ILAL) [Asp98, AR02, Ter01, Ter07].

Recall that SRN is a recursion theoretical system. It is generated from a set of basic functions, closed under the *safe composition* and *safe recursion* schemes. SRN captures FP by partitioning the set of arguments of each function $g(\vec{n}, \vec{s})$ into those that are *normal*, namely $\vec{n}$, and those that are *safe*, *i.e.* $\vec{s}$. The basic functions can only have safe arguments. The crucial features of a function f, defined by an instance of the safe recursive scheme, are: (i) the unfolding of f is driven by a normal argument, and (ii) the recursive call of f may only appear in a safe argument position, as far as the unfolding proceeds, ensuring that recursion over the result of a function defined by recursion is not possible. Section 4 formally recalls SRN. In fact, it also recalls *Composition-linear safe recursion on notation* (ClSRN), already introduced in [Rov07] where, however, it was called as *Quasi-linear safe recursion on notation*. ClSRN restricts SRN; It is defined on a set of basic functions, closed under the *full* safe recursion scheme of SRN and a *linear safe composition scheme* that uses linearly its safe variables. Namely, ClSRN strictly generalizes BC$^-$ [MO04]. Recall that BC$^-$ is SRN where *both* the safe composition and recursion schemes can exclusively use their safe variables *linearly*.

The reason to recall ClSRN here is the following one. We already know that ClSRN can be embedded into WALT [Rov07]. By using that result, we show that, in fact, *full* SRN can be embedded into WALT, so relating "predicative recursion" to "stratification" without any restriction.

Formally, the relation reads as follows. There exists an interpretation map $[\![\ ]\!]$ from SRN to WALT, such that, for every $f(n_1, \ldots, n_k, s_1, \ldots, s_l) \in$ SRN, with k *normal* and l *safe* arguments, we can prove that: (i) if $f(n_1, \ldots, n_k, s_1, \ldots, s_l) = n$, then $[\![f(n_1, \ldots, n_k, s_1, \ldots, s_l)]\!]$ reduces to $[\![n]\!]$, using $\rightsquigarrow_w$, a rewriting relation which is a mix of both call-by-name and call-by-value β-reductions, and (ii) $[\![f(n_1, \ldots, n_k, s_1, \ldots, s_l)]\!]$ has type $\$^m\mathbf{W}$, since $[\![f]\!]$ has type $\overbrace{\$\mathbf{W} \multimap \ldots \multimap \$\mathbf{W}}^{k} \multimap \overbrace{\$^m\mathbf{W} \multimap \ldots \multimap \$^m\mathbf{W}}^{l} \multimap \$^m\mathbf{W}$, for some $m \geq 1$, the type $\mathbf{W}$ being the one for binary words in WALT.

Point (i) shows that WALT is SRN-*complete*. Namely, WALT is the first system, derived as a restriction of Linear logic to characterize FP, where *full* SRN can be embedded.

Point (ii) links m to the complexity of the definition of f: m depends on the number of nested safe compositions and of safe recursive schemes that define f. The types explicitly show a layered structure inside the normal and safe arguments of SRN: the type of a safe argument is $m \geq 1$ $\$$-modality occurrences deep because a safe argument can be used in the course of a recursive unfolding to produce a result. Orthogonally, the depth of the type of every normal argument is limited to 1, so giving to every normal argument the necessary "replication power", required to duplicate syntactic structure in the course of an unfolding. On one side, this means that the Light linear logic-like systems say that the weaker is the possibility of a word to replicate structure, behaving it as an iterator, the deeper is its type. On the other, the recursive systems like SRN are based exactly on the reversed idea, though this cannot be formally stated in terms of any typing information inside SRN.

A further consequence of embedding SRN into WALT is that we obtain a second version of such a proof, the first being in [Rov07]. Recall that "polytime completeness" means that every polynomial Turing machine can be represented as a term of WALT.

However, a more relevant consequence of looking for the formal relations between two systems like SRN and WALT, as we have just done, is the way we prove the SRN-completeness of WALT. We shall see that it is obtained by *simulating the full composition scheme* of SRN through the *linear* safe composition scheme of ClSRN and its *full* recursive scheme, which coincides to the one of SRN. This candidates ClSRN to be *a linear kernel of* SRN, so pointing to the existence of a reformulation of SRN itself in terms of more primitive and linear constructs, as we shall discuss in the conclusions.

Finally, we observe that WALT yields a higher-order characterization of FP in the lines of Higher type ramified recursion (HTRR) [BNS00] and Higher order linear ramified recursion (HOLRR) [DLMR04], but with a relevant difference. Both HTRR and HOLRR build their terms by assuming the existence of constant symbols, like words, successors, *etc.*. On the contrary, no constant symbol is used inside WALT where everything is defined from scratch, exploiting its II-order structural proof-theoretic roots.

Outline. Section 2 intuitively recalls the main intuitions about WALT, by pointing out how it weakens the design principles of ILAL. Section 3 recalls the technical parts of WALT, required to program the combinators that allow to represent the full composition scheme of SRN in WALT, as shown in Section 5, so yielding the SRN-completeness. Section 4 formally recalls SRN in the style of [BW96]. Section 7 delineates some possible research directions.

2 Overviewing WALT intuitively

The full technical introduction of WALT is [Rov07].

Here we want to recall the key ideas about WALT at an intuitive level. The main goal is to illustrate the main reasons why WALT is more expressive than other deductive systems, derived from Linear logic, to characterize the class of polynomial functions (FP).

Squaring chains. WALT contains the main "complex" structure of Intuitionistic light affine logic (ILAL) which we call *squaring chain*. A graph representation of an instance of squaring chain is in Figure 1.(a). On

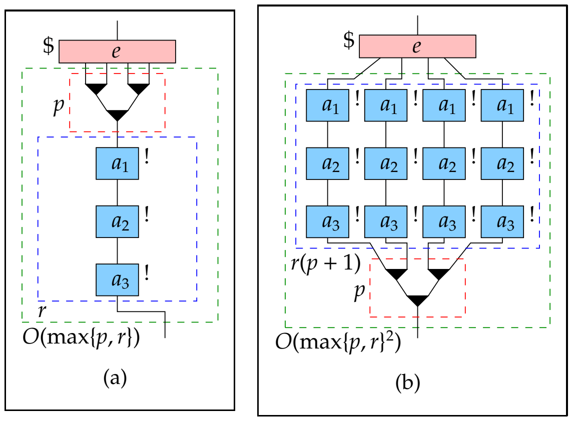


Figure 1: A squaring chain and its reduct

top of it there is a tree of nodes, all the black triangles, that contract a set of assumptions of the given topmost $-box e. Below the tree of contraction nodes there is a list of !-boxes a_1, a_2, a_3. Every of them depends on at most a single assumption, which is the basic constraint of !-boxes of ILAL. Of course, generally, the number of contractions nodes and of !-boxes in a squaring chain is arbitrary and the tree they form need not to be perfectly balanced. The chain is dubbed as "squaring" because its normalization leads to the configuration in Figure 1.(b), where the size $O(\max\{p,r\})$, essentially, "squares" to a value which is $O(\max\{p,r\}^2)$.

Weak squaring and stuck chains. However, besides the squaring chains, WALT contains both *weak squaring chains* and *stuck chains*, and its expressive power relies on their existence. A graph representation of both types of chains is in Figure 2.(a). The one to the left is a weak squaring chain. The other, to the right, is a stuck chain.

The weak squaring chain contains a tree of contraction nodes. However, it is based on a more liberalized form of !-boxes. They may depend on more than one assumption: one of them must have a !-modal type, the others a $-modal type. The chain under description is "weak squaring" because *only after* we merge the $-boxes c_1, c_2 into the !-boxes a_1, a_2, respectively, it transforms to a squaring chain, with !-boxes $a_1 \bowtie c_1, a_2 \bowtie c_2, a_3$, that can be squared to the configuration to the left in Figure 3. We insist on observing that, before the merging of boxes, no squaring can occur.

This is why the configuration to the right in Figure 2.(b) is a stuck chain, and not a weak squaring one. Its "squaring through normalization" cannot start, even if we merge b_2 and d_2, because there is no $-box plugged into the assumption of type $\$A$ of the !-box b_1. So, the chain to the right in Figure 2.(b) is stuck until the context, eventually, supplies a closed $-box with conclusion $\$A$ that can be merged into b_1, so yielding a squaring chain that we can "square".

We conclude by remarking that the form of weak squaring and stuck chains is more general than the one in the given example. Indeed, not only a single closed $-box can be dangling down the $-modal assumptions

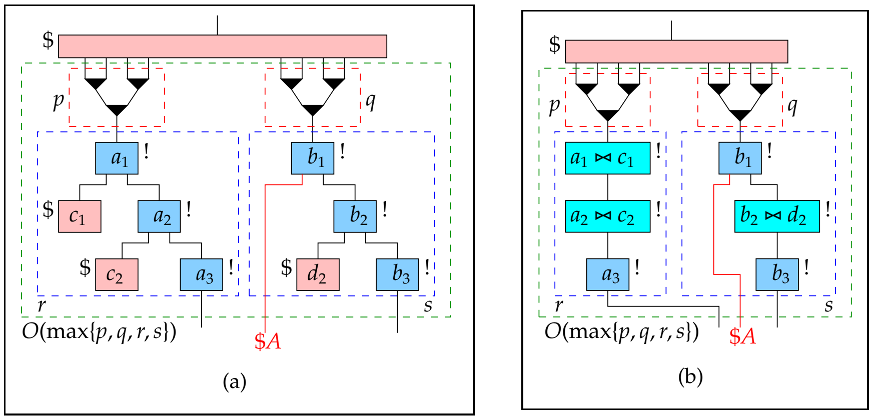


Figure 2: A weak squaring and a stuck chain

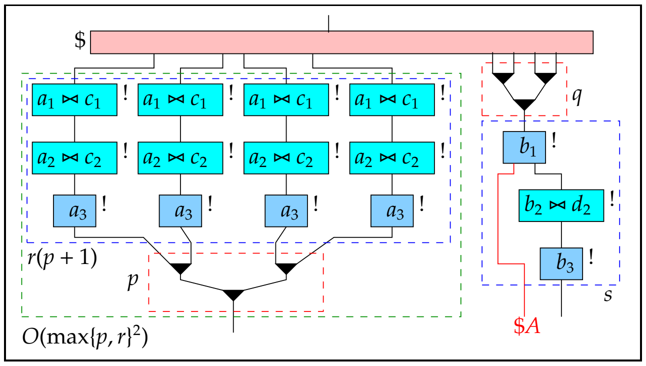


Figure 3: The reduct of a weak squaring chain

135 of a !-box, but there can be a whole tree whose nodes can only be \$-boxes, which must be closed whenever they constitute the leaves of the tree itself.

The lazy nature of WALT. WALT induces a call-by-value dynamics on the λ-terms it gives types to as consequence of the more general form of its !-boxes, as compared to ILAL. As we have seen, a chain is stuck until the context supplies the \$-boxes that close all the assumptions, with \$-modal type, of those !-boxes
140 that must be duplicated. All such assumptions are used to represent the parameters in the simulation of the full recursive scheme of SRN inside WALT, with the right type. We shall recall the main idea under some simplifying assumptions, to keep things more readable.

Let us assume to have a function f, recursively defined as $f(0, a) = g(a)$, and $f(n, a) = h(n-1, a, f(n-1, a))$, with $n \geq 1$. We want to show how simulating its top-down recursive unfolding:

$$f(n, a) = h(n-1, a, f(n-1, a)) = \ldots = h(n-1, a, h(n-2, a, \ldots h(0, a, g(a)) \ldots))$$

by a bottom-up reconstruction that iterates some transition functions on suitable configurations and pre-configurations. The reconstruction requires to assume H, G be the interpretations of h, g, respectively, in WALT. Moreover, we assume the unary strings $\overline{n}, \overline{a}$, not words, represent n, a in WALT. What we are going to say, though, keeps holding with f of arbitrary arity and with words as its arguments, instead of strings. In WALT we can develop sequences of computations like the following one, where all the terms can be correctly typed:

$$\langle\!\langle G\overline{a}, [\underbrace{\overline{0},\ldots,\overline{0}}_{n+1}], [\underbrace{\overline{a},\ldots,\overline{a}}_{n+1}]\rangle\!\rangle \rightsquigarrow^*_w \tag{1}$$

$$\langle\!\langle G\overline{a}, \langle \overline{0}, [\underbrace{\overline{1},\ldots,\overline{1}}_{n}]\rangle, \langle \overline{a}, [\underbrace{\overline{a},\ldots,\overline{a}}_{n}]\rangle\rangle\!\rangle \rightsquigarrow^*_w \langle\!\langle H\,\overline{0}\,\overline{a}\,(G\overline{a}), [\underbrace{\overline{1},\ldots,\overline{1}}_{n}], [\underbrace{\overline{a},\ldots,\overline{a}}_{n}]\rangle\!\rangle \rightsquigarrow^*_w \tag{2}$$

$$\langle\!\langle H\,\overline{0}\,\overline{a}\,(G\overline{a}), \langle \overline{1}, [\underbrace{\overline{2},\ldots,\overline{2}}_{n-1}]\rangle, \langle \overline{a}, [\underbrace{\overline{a},\ldots,\overline{a}}_{n-1}]\rangle\rangle\!\rangle \rightsquigarrow^*_w \langle\!\langle H\,\overline{1}\,\overline{a}\,(H\,\overline{0}\,\overline{a}\,(G\overline{a})), [\underbrace{\overline{1},\ldots,\overline{1}}_{n-1}], [\underbrace{\overline{a},\ldots,\overline{a}}_{n-1}]\rangle\!\rangle \rightsquigarrow^*_w \ldots \tag{3}$$

The ideal column to the right of $\rightsquigarrow^*_w$ contains *configurations*, the topmost being the *initial* one. The column to the left of $\rightsquigarrow^*_w$ contains *pre-configurations*. Every pre-configuration comes from its preceding configuration by (i) separating head and tail of every list, and storing them as the two components of a same pair, (ii) only on the leftmost list, simultaneously to the separation, the successor is mapped on the tail.

Every configuration, other than the initial one, is obtained from its preceding pre-configuration by the application of an instance of H to the first element of every pair, and to the first element of the whole pre-configuration, which accumulates the partial result of the bottom-up reconstruction.

The *main point* for everything to work correctly in the above simulation is to produce $[\underbrace{\overline{a},\ldots,\overline{a}}_{n+1}]$ with the right type. This is obtained by using the term in Figure 4 whose definition is substantially based on an instance of the more general !-box existing in WALT, but not in ILAL. The assumption of type $\$\mathbf{N}$ in the !-box waits for $\overline{a}$, one \$-box deep. As soon as this value is supplied, the !-box is ready for the duplication by means of the contraction nodes that may be contained in the Church numeral $\overline{n}$ of type $\forall\alpha.!(\alpha \multimap \alpha) \multimap \$(\alpha \multimap \alpha)$, which is the second argument of the whole term. Once both $\overline{a}$ and $\overline{n}$ have been given, the result of the whole term is a list of copies of $\overline{a}$, whose type is the one we can expect: $\forall\alpha.!(\$\mathbf{N} \multimap \alpha \multimap \alpha) \multimap \$(\alpha \multimap \alpha)$, that we shorten as $\mathbf{LN}$. The use of the assumption with type $\$\mathbf{N}$ in the !-box is the key step to obtain a result of type $\mathbf{LN}$ which, somewhat, absorbs the \$-box initially around the parameter of $\overline{a}$. Without this merging we could not obtain a representation of the iterator whose safe arguments are at the same depth as the result, as required to represent the full recursion scheme of SRN in WALT.

The full SRN-composition scheme in WALT. Once the full recursion scheme of SRN is at hand in WALT, we can use it to encode also the full composition scheme of SRN. Figure 5 shows an example of functional block diagram that summarizes how the full composition scheme of SRN becomes a term of WALT. Let us assume F, G, H_1, H_2 be terms of WALT that represent the SRN functions f, g, h_1, h_2, respectively, and that we need to compose as follows. f takes one normal and two safe arguments which are supplied by g, that depends on a single safe argument, and by h_1, h_2. Also, we assume that both h_1, h_2 require a single normal argument, but h_1 needs three safe arguments, while h_2 only one. The first operation to represent the full composition scheme of SRN in WALT is to generate the terms F, G, H_1, H_2. H_1 will be obtained from h_1 as the result of an inductive translation, as we might expect. H_2 will be defined from h_2 with the same inductive process. However, this would lead to a term with safe arity 1. To obtain a term H_2 with safe arity 3, we extend the resulting term to erase two of its three safe arguments: H_2 and the two bullets close to it in Figure 5 represent such a final term. The same holds for F which must erase its third safe argument. Notice that the safe value it erases is supplied by the dummy function $\overline{0}\bullet\bullet\bullet\bullet$, which is constantly equal to $\overline{0}$, after the erasure of all its arguments: one normal, the others safe. G, supplying it the normal value to F, does not present any problem. The translation process of the functions being composed, all with the same safe arity, occurs inside the square composition $\boxdot^{1;3}_1$: the topmost parameter 1 is the normal arity of every of the terms being composed, 3 is the maximal safe arity, namely, the value with respect to which we normalize the terms we generate, and the lowermost 1 is the normal arity of F. So, $\boxdot^{1;3}_1[F, G, H_1, H_2]$ has normal arity 1 and safe arity 9, since every of the three composed terms will have safe arity 3. After the normalized composition, the ideal functional block χ rearranges the safe arguments: the first safe arguments of $H_1, H_2\bullet\bullet$, and $\overline{\overline{0}}\bullet\bullet\bullet\bullet$ are put one closed to the others, and the same is done for the second and third ones. The goal is to share

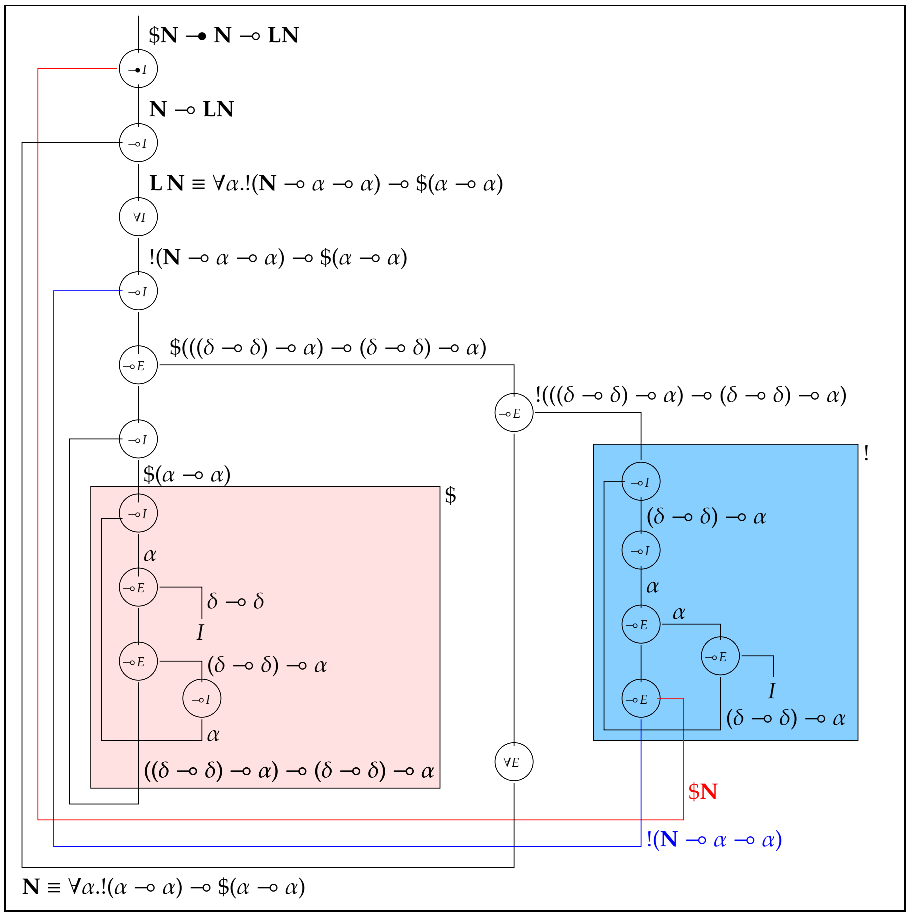

$N ⊸• N ⊸ LN
N ⊸ LN
L N ≡ ∀α.!(N ⊸ α ⊸ α) ⊸ $(α ⊸ α)
!(N ⊸ α ⊸ α) ⊸ $(α ⊸ α)
$(((δ ⊸ δ) ⊸ α) ⊸ (δ ⊸ δ) ⊸ α)
!(((δ ⊸ δ) ⊸ α) ⊸ (δ ⊸ δ) ⊸ α)
$(α ⊸ α)
$
!
α
δ ⊸ δ
I
(δ ⊸ δ) ⊸ α
α
((δ ⊸ δ) ⊸ α) ⊸ (δ ⊸ δ) ⊸ α
(δ ⊸ δ) ⊸ α
α
α
I
(δ ⊸ δ) ⊸ α
$N
!(N ⊸ α ⊸ α)
N ≡ ∀α.!(α ⊸ α) ⊸ $(α ⊸ α)

Figure 4: The term that generates the list of constants in a recursive scheme

each of the group into a single safe argument. This happens inside the functional block $\curlyvee_m^{1;3\backslash 3}$ which applies, one after the other, three further blocks $\circlearrowleft\mathtt{mY}_m^{1;(6,3)}$, $\circlearrowleft\mathtt{mY}_m^{1;(4,3)}$, $\circlearrowleft\mathtt{mY}_m^{1;(2,3)}$. The behavior of every of these blocks is to share three safe arguments in input into a single safe argument, and to rotate them so that a new group of safe arguments gets ready for the sharing of its components by means of the subsequent block. The sharing of the safe arguments is hidden inside the black triangles. Every of them contains two one-step long iterations that share the same safe value in the last two positions of a given term M. The following unfolding illustrates the idea about the behavior of one of such one-step long iterations:

$$\begin{aligned} \mathtt{Y}_m^{1;3}[M]\, n_1\, s_1\, s_2\, s_3 &= M\, n_1\, s_1\, s_2\, s_3\, (\mathtt{Y}_m^{1;3}[M]\, n_1\, s_1\, s_2\, s_3) \\ &= M\, n_1\, s_1\, s_2\, s_3\, ((\backslash x_1\, y_1\, y_2\, y_3.y_3)\, n_1\, s_1\, s_2\, s_3) \\ &= M\, n_1\, s_1\, s_2\, s_3\, s_3 \end{aligned} \tag{4}$$

We insist remarking that (4) gives only the idea of what happens. The definition of $\mathtt{Y}_m^{1;3}$ is based on the iteration term, typeable in WALT, and will be formally given in Section 5, together with its dynamics. Once the safe arguments of $\boxdot_1^{1;3}[F, G, H_1, H_2]$ have been shared we are left with a term waiting for three safe and one normal arguments. The latter is replicated four times by ∇_4^1 that, standardly, iterates a tuple of successors, starting from a tuple of four instances of $\overline{\overline{0}}$. Just to remark it again, the above translation mechanism, can be set for any normal and safe arities of any number of functions in SRN.

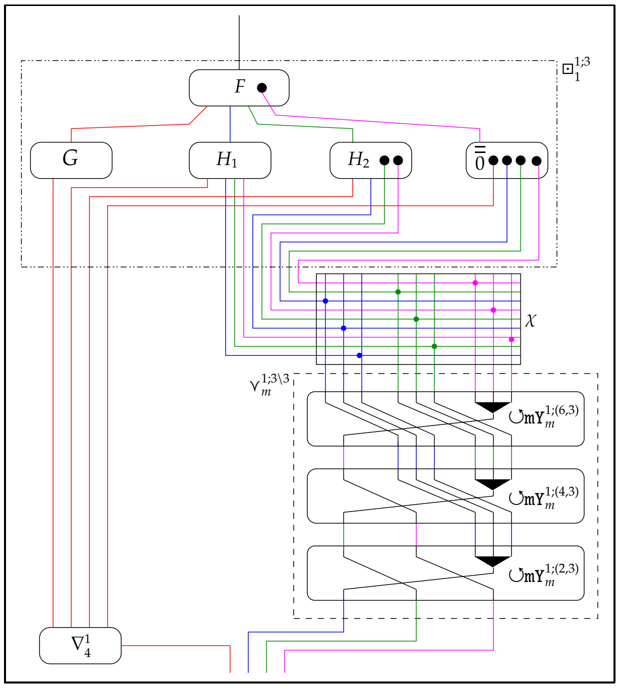

F
G
H_1
H_2
$\overline{\overline{0}}$
$\boxdot_1^{1;3}$
χ
$\curlyvee_m^{1;3\backslash 3}$
$\circlearrowleft \mathtt{mY}_m^{1;(6,3)}$
$\circlearrowleft \mathtt{mY}_m^{1;(4,3)}$
$\circlearrowleft \mathtt{mY}_m^{1;(2,3)}$
∇_4^1

Figure 5: Functional block scheme of SRN-composition in WALT

3 Overviewing WALT technically

Here we recall the main aspects of Weak Affine Light Typing (WALT), as developed in [Rov07], and which are required to present our results. Recall that WALT is a type assignment for λ-terms.

The λ-terms. The λ-terms are generated by the grammar $M ::= x \mid (\backslash x.M) \mid (MM)$, where x belongs to a countable set of λ-variables. An abstraction $\backslash x.M$ binds the (free) occurrence of x in M. Given a term M, the set of its free variables, those ones which are not bound, is $\mathrm{FV}(M)$. A *closed term* has no free variables. The *cardinality of a free variable in a term* is $\mathrm{no}(x, M)$ and counts the number of *free occurrences of x in M*:

$$\begin{array}{rlr} \mathrm{no}(x,x) = 1 & \mathrm{no}(x,y) = 0 & (x \not\equiv y) \\ \mathrm{no}(x,\backslash x.M) = 0 & \mathrm{no}(x,\backslash y.M) = \mathrm{no}(x,M) & (x \not\equiv y) \\ \mathrm{no}(x,MN) = \mathrm{no}(x,M) + \mathrm{no}(x,N) & & \end{array}$$

$M\{^{N_1}/_{x_1} \cdots {}^{N_m}/_{x_m}\}$ denotes the usual capture free simultaneous substitution of every N_i for the corresponding x_i, with $1 \leq i \leq m$. Parentheses are left-associative, so $((\cdots((MM)M)\cdots)M)$ shortens to $MMM\cdots M$. A sequence of abstractions $(\backslash x_1.\ldots(\backslash x_m.M)\ldots)$ is shortened by $\backslash x_1\ldots x_m.M$, for any m. Λ_{V} is the set of the λ-terms which are *values*, generated by $V ::= x \mid (\backslash x.M)$, where M is any λ-term. The *size of a term* $|M|$ gives the *dimension* of M as expected: $|x| = 1, |\backslash x.M| = |M| + 1, |MN| = |M| + |N| + 1$.

The types of WALT. They are formulae that belong to the language, generated by the following grammar:

$$\begin{aligned} A &::= L \mid !A \mid \$A \\ L &::= \alpha \mid A \multimap A \mid \$A \multimap\!\!\bullet\, A \mid \forall\alpha.L \end{aligned}$$

A is the start symbol. A *modal* formula has form $!A$ or $\$A$, and, in particular, $!A$ is !-modal, while $\$A$ is \$-modal. L generates *linear*, or *non modal*, formulae, which are ranged over by L, L'. Generic formulae by A, B, C, instead. Notice that the substitution of α in $\forall\alpha.L$ for a linear type L', produces $L\{^{L'}/_{\alpha}\}$ which is still linear. Somewhat conversely, a universal quantification cannot hide a modal type.

The rules of WALT. Figure 6 gives the deductive rules of WALT, which deduce judgments $\Gamma;\Delta;\mathcal{E} \vdash M : A$. Γ and Δ are sets of type assignments, namely of pairs $x : A$. $\mathcal{E}$ is a set of pairs $(\Theta;\Phi)$ such that both Θ and Φ are sets of type assignments as well.

$\mathrm{Dom}(\{x_1 : A_1, \ldots, x_n : A_n\}) = \{x_1, \ldots, x_n\}$ is the *domain* of any set $\{x_1 : A_1, \ldots, x_n : A_n\}$ of type assignments. Γ will denote a set of *linear type assignments* $x : L$. Every $x \in \mathrm{Dom}(\Gamma)$ is called *linear variable*. Δ will denote a set of *linear partially discharged* type assignments. Every $x \in \mathrm{Dom}(\Delta)$ is called *linear partially discharged*. $\mathcal{E}$ will denote a set of *partially discharged contexts*. $\mathcal{E}$ can be either empty or it contains pairs $(\Theta_1;\Phi_1), \ldots, (\Theta_n;\Phi_n)$ where, for every $i \in \{1, \ldots, n\}$, the following four points hold: (i) Θ_i is a set of *elementary partially discharged* type assignment. Every $x \in \mathrm{Dom}(\Theta_i)$ is called *elementary*; (ii) Φ_i is either empty or it is a singleton $x : A$. We call *x polynomial*; (iii) only one between $\Phi_1, \ldots, \Phi_n$ can be $\emptyset$; (iv) the domains of any two Φ_i and Φ_j are distinct, whenever $i \neq j$.

For every $\mathcal{E} = \bigcup_{i=1}^{n}\{(\Theta_i;\Phi_i)\}$, $\mathrm{Dom}(\mathcal{E})$ is $(\bigcup_{i=1}^{n} \mathrm{Dom}(\Theta_i)) \cup (\bigcup_{i=1}^{n} \mathrm{Dom}(\Phi_i))$. In every of the rules of WALT the domain of two sets of type assignments Φ_M and Φ_N may intersect when Φ_M and Φ_N are part of two partially discharged contexts $\mathcal{E}_M$ and $\mathcal{E}_N$ that belong to two distinct premises of a rule. This observation justifies the definition of $\mathcal{E}_M \sqcup \mathcal{E}_N$ that merges $\mathcal{E}_M$ and $\mathcal{E}_N$, preserving the structure of a partially discharged context:

$$\begin{aligned} &\mathcal{E}_M \sqcup \mathcal{E}_N = \\ &\quad \{(\Theta_M, \Theta_N;\Phi) \mid (\Theta_M;\Phi) \in \mathcal{E}_M \text{ and } (\Theta_N;\Phi) \in \mathcal{E}_N\} \cup \\ &\quad \{(\Theta_M;\Phi_M) \mid (\Theta_M;\Phi_M) \in \mathcal{E}_M \text{ and there is no } \Theta_N \text{ such that } (\Theta_N;\Phi_M) \text{ in } \mathcal{E}_N\} \cup \\ &\quad \{(\Theta_N;\Phi_N) \mid (\Theta_N;\Phi_N) \in \mathcal{E}_N \text{ and there is no } \Theta_M \text{ such that } (\Theta_M;\Phi_N) \text{ in } \mathcal{E}_M\} \end{aligned}$$

The sequence $\mathcal{E}, (\Theta;\Phi)$ denotes that $(\Theta;\Phi) \notin \mathcal{E}$. Also, $\mathcal{E} \sqcup \{(\emptyset;\emptyset)\} = \mathcal{E} \sqcup \emptyset = \mathcal{E}$. In every other cases, the domain of two sets of type assignments that belong to two distinct premises of a rule of WALT *must* be disjoint. Λ^{T} is

$$\frac{}{\Gamma, x\!:\!L; \Delta; \mathcal{E} \vdash x\!:\!L}\ A$$

$$\frac{\Gamma; \Delta; \mathcal{E}, (\Theta_x; \{x\!:\!A\}), (\Theta_y; \{y\!:\!A\}) \vdash M\!:\!B}{\Gamma; \Delta; \mathcal{E} \sqcup \{(\Theta_x, \Theta_y; \{z\!:\!A\})\} \vdash M\{^z/_x{}^z/_y\}\!:\!B}\ C$$

$$\frac{\Gamma, x\!:\!L; \Delta; \mathcal{E} \vdash M\!:\!B}{\Gamma; \Delta; \mathcal{E} \vdash \backslash x.M\!:\!L \multimap B}\ \multimap I \qquad \frac{\Gamma; \Delta, x\!:\!A; \mathcal{E} \vdash M\!:\!B}{\Gamma; \Delta; \mathcal{E} \vdash \backslash x.M\!:\!\$A \multimap B}\ \multimap I_{\$}$$

$$\frac{\Gamma_M; \Delta_M; \mathcal{E}_M \vdash M\!:\!A \multimap B \quad \Gamma_N; \Delta_N; \mathcal{E}_N \vdash N\!:\!A \quad A \not\equiv !C,\ \text{for any}\ C}{\Gamma_M, \Gamma_N; \Delta_M, \Delta_N; \mathcal{E}_M \sqcup \mathcal{E}_N \vdash MN\!:\!B}\ \multimap E$$

$$\frac{\Gamma; \Delta; \mathcal{E}, (\Theta; \{x\!:\!A\}) \vdash M\!:\!B}{\Gamma; \Delta; \mathcal{E} \sqcup \{(\Theta; \emptyset)\} \vdash \backslash x.M\!:\!!A \multimap B}\ \multimap I_{!}$$

$$\frac{\Gamma_M; \Delta_M; \mathcal{E}_M \vdash M\!:\!!A \multimap B \quad \Gamma_N; \Delta_N; \mathcal{E}_N \vdash N\!:\!!A \quad \mathcal{E}_M \subseteq \{(\emptyset; \Phi_1), \ldots, (\emptyset; \Phi_n)\}}{\Gamma_M, \Gamma_N; \Delta_M, \Delta_N; \mathcal{E}_M \sqcup \mathcal{E}_N \vdash MN\!:\!B}\ \multimap E_{!}$$

$$\frac{\Gamma; \Delta; \mathcal{E}, (\Theta, x\!:\!A; \emptyset) \vdash M\!:\!B}{\Gamma; \Delta; \mathcal{E} \sqcup \{(\Theta; \emptyset)\} \vdash \backslash x.M\!:\!\$A \mathrel{-\!\!\bullet} B}\ \mathrel{-\!\!\bullet} I$$

$$\frac{\Gamma_M; \Delta; \mathcal{E}_M \vdash M\!:\!\$A \mathrel{-\!\!\bullet} B \quad \emptyset; \emptyset; \mathcal{E}_N \vdash N\!:\!\$A \quad \mathcal{E}_N \subseteq \{(\Theta; \emptyset)\}}{\Gamma_M; \Delta; \mathcal{E}_M \sqcup \mathcal{E}_N \vdash MN\!:\!B}\ \mathrel{-\!\!\bullet} E$$

$$\frac{\Gamma; \Delta'; \{(\Theta'; \emptyset)\} \vdash M\!:\!B \quad \Gamma \subseteq \Delta \cup \bigcup_{i=1}^{m} \Theta_i \cup \bigcup_{i=1}^{m} \Phi_i \quad \Theta_i \neq \emptyset \text{ iff } \Phi_i = \emptyset}{\Gamma'; \$\Delta', \Delta; \{(\$\Theta'; \emptyset)\} \sqcup \{(\Theta_1; \Phi_1)\} \sqcup \ldots \sqcup \{(\Theta_m; \Phi_m)\} \vdash M\!:\!\$B}\ \$$$

$$\frac{\Gamma; \emptyset; \{(\Theta'; \emptyset)\} \vdash M\!:\!B \quad \Gamma \subseteq \Theta \cup \Phi \quad \Theta \neq \emptyset \Rightarrow \mathrm{Dom}(\Phi) \cap \mathrm{FV}(M) \neq \emptyset}{\Gamma'; \Delta; \{(\$\Theta'; \emptyset)\} \sqcup \{(\Theta; \Phi)\} \vdash M\!:\!!B}\ !$$

$$\frac{\Gamma; \Delta; \mathcal{E} \vdash M\!:\!L \quad \alpha \text{ not free in } \Gamma, \Delta \text{ and } \mathcal{E}}{\Gamma; \Delta; \mathcal{E} \vdash M\!:\!\forall\alpha.L}\ \forall I \qquad \frac{\Gamma; \Delta; \mathcal{E} \vdash M\!:\!\forall\alpha.L}{\Gamma; \Delta; \mathcal{E} \vdash M\!:\!L\{L'/\alpha\}}\ \forall E$$

Figure 6: Weak Affine Light Typing

the subset of *typeable* elements M of Λ, namely, those for which a deduction Π with conclusion $\Gamma;\Delta;\mathcal{E} \vdash M\!:\!A$ exists, denoted by $\Pi \rhd \Gamma;\Delta;\mathcal{E} \vdash M\!:\!A$.

WALT and System F. WALT is a subsystem of System F. This means that if $\Gamma;\Delta;\mathcal{E} \vdash M\!:\!A$ then M has type $t(A)$ from the set of assumptions $T(\Gamma;\Delta;\mathcal{E})$ in System F, where:

$$\begin{array}{ccc} t(\alpha) = \alpha & t(A \multimap B) = t(A) \rightarrow t(B) & t(\$A) = t(A) \\ t(\forall\alpha.A) = \forall\alpha.t(A) & t(A \multimapdot B) = t(A) \rightarrow t(B) & t(!A) = t(A) \end{array}$$

and T is the obvious extension of the map t to the types in $\Gamma;\Delta;\mathcal{E}$.

3.1 On the rules of WALT

The bound on the number of normalization steps of any deduction of WALT is a consequence of the stratified nature that WALT inherits from ILAL. "Stratification" means that every deduction Π of WALT can be thought of as it was organized into levels, so that the logical rules of Π may be at different depths. The normalization preserves the levels: if an instance of a rule R in Π is at depth d, then it will keep being at depth d after any number of normalization steps that, of course, do not erase it. The only duplication allowed is of deductions Π that have undergone an instance r of the ! rule, namely the conclusion of Π has a !-modal type, introduced by r. Ideally, the ! rule defines a, so called, !-box around the deduction that proves its premise. Figure 7 shows, side by side, a canonical instance of the rule ! and the !-box that would correspond to it if we imagined to associate a proof net notation to the derivations of WALT. The hypothesis is that Π_M, with conclusion of

$$\frac{x_1\!:\!C_1,\ldots,x_n\!:\!C_n,x\!:\!B;\emptyset;\emptyset \vdash M\!:\!A \quad n>0 \Rightarrow x \in \mathrm{FV}(M)}{\emptyset;\emptyset;\{(\{x_1\!:\!C_1,\ldots,x_n\!:\!C_n\};\{x\!:\!B\})\} \vdash M\!:\!!A}\ !$$

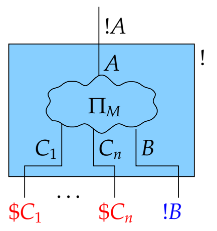


Figure 7: The canonical instance of !-box/rule in WALT

type A and assumptions $C_1,\ldots,C_n,B$, corresponds to the term M that has type B from the set of linear type assignments $x_1\!:\!C_1,\ldots,x_n\!:\!C_n,x\!:\!B$. The application of the rule ! corresponds to putting the !-box around Π_M.

The condition $n > 0 \Rightarrow x \in \mathrm{FV}(M)$ assures that the substitution of some closed term N, with type $!B$, for x in M, cannot yield $M\{^N/_x\}$ that only depends on a single assumption. Namely, we want to avoid that a sequence of normalization steps, can yield a judgment $\emptyset;\emptyset;\{(\{x_1\!:\!C_1\};\emptyset)\} \vdash M'\!:\!!A$, where $!A$ says that it can be duplicated, but whose free assumption says that it cannot, since WALT does not have the contraction on \$-modal assumptions.

We remark that the !-box can be put around a derivation Π that may depend on more than one assumption, letting WALT be a strict generalization of ILAL, whose !-boxes, in the context of WALT, take the form:

$$\frac{\Phi;\emptyset;\emptyset \vdash M\!:\!B \quad \Phi \subseteq \{x\!:\!A\}}{\emptyset;\emptyset;\{(\emptyset;\Phi)\} \vdash M\!:\!!B}$$

The elementary partially discharged assumptions the generalized !-boxes may depend on can only be replaced, in the course of the normalization steps, by the conclusion of \$-boxes exclusively depending on elementary partially discharged assumptions as well. Figure 8 shows, with the help of a net, that such

$$\dfrac{\begin{array}{l}\Gamma;\Delta;\mathcal{E}_M \vdash M:\$C \mathrel{-\!\!\bullet} A\\ \emptyset;\emptyset;\mathcal{E}_N \vdash N:\$C\\ \mathcal{E}_N \subseteq \{(\Theta;\emptyset)\}\end{array}}{\Gamma;\Delta;\mathcal{E}_M \sqcup \mathcal{E}_N \vdash MN:D} \mathrel{-\!\!\bullet} E$$

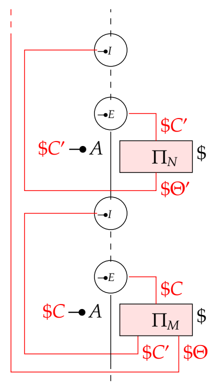
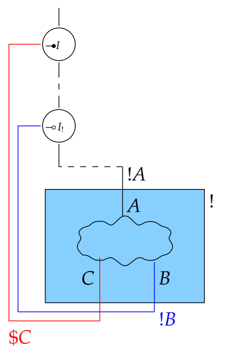

$\mathrel{-\!\!\bullet} I$
$\mathrel{-\!\!\bullet} E$
$\$C'$
$\$C' \mathrel{-\!\!\bullet} A$
Π_N
$\$$
$\$\Theta'$
$\mathrel{-\!\!\bullet} I$
$\mathrel{-\!\!\bullet} E$
$\$C$
$\$C \mathrel{-\!\!\bullet} A$
Π_M
$\$$
$\$C'$
$\$\Theta$

Figure 8: "Net" meaning of the rule $\mathrel{-\!\!\bullet} E$ in WALT



$$\dfrac{\Gamma;\Delta;\mathcal{E},(\Theta;\{x:B\}) \vdash M:A}{\Gamma;\Delta;\mathcal{E} \sqcup \{(\Theta;\emptyset)\} \vdash \backslash x.M:\ !B \multimap A} \multimap I_!$$

$$\dfrac{\Gamma;\Delta;\mathcal{E},(\{\Theta,y:C\};\emptyset) \vdash M:A}{\Gamma;\Delta;\mathcal{E},(\Theta;\emptyset) \vdash \backslash x.M:\$C \mathrel{-\!\!\bullet} A} \mathrel{-\!\!\bullet} I$$

Figure 9: "Net" meaning of the rules $\mathrel{-\!\!\bullet} I, \multimap I_!$ in WALT

a behavior is a consequence of restricting $\mathcal{E}_N$, in the rule $\mathrel{-\!\!\bullet} E$, to the form $\{(\Theta;\emptyset)\}$. The rule $\mathrel{-\!\!\bullet} E$ comes with $\mathrel{-\!\!\bullet} I$ that can discharge elementary partially discharged assumptions only when the corresponding polynomial assumption has already been discharged by $\multimap I_!$, as in Figure 9. The net in such a figure shows the mandatory discharging order. Finally, Figure 10 shows, with the help of a net, how $\multimap E_!$ consistently forces the application of some given M of type $!B \multimap C$ to an N, of type $!B$, according to an order which reverses the one we must use to apply $\multimap I_!$, and $\mathrel{-\!\!\bullet} I$.

Summing up, WALT allows to type λ-terms more liberally than ILAL, while keeping the same normalization principles: the stratification is never canceled, and only deductions that, eventually, depend on at most one free variable may be effectively duplicated as effect of the normalization. This is why WALT does not enjoy a full normalizing procedure, the analogous of the cut elimination for a corresponding sequent calculus formulation, as the coming section recalls.

3.2 The dynamics of WALT.

Recall that the *call-by-name*, or *lazy*, β-reduction on the λ-terms is the contextual closure of rewriting relation $(\backslash x.M)N \to_{\mathrm{n}} M\{^N/_x\}$. The *call-by-value*, or *eager*, β-reduction, instead, is the contextual closure of $(\backslash x.M)N \to_{\mathrm{v}} M\{^N/_x\}$, where $N \in \Lambda_{\mathrm{V}}$.

The subject reduction of the rules in Figure 6 holds only on the following restriction $\rightsquigarrow_w$ of $\to_{\mathrm{n}} \cup \to_{\mathrm{v}}$:

$$\frac{\begin{array}{l}\Gamma_M;\Delta_M;\mathcal{E}_M \vdash M\colon !B \multimap C\\ \Gamma_N;\Delta_N;\mathcal{E}_N \vdash N\colon !B\\ \mathcal{E}_M \subseteq \{(\emptyset;\Phi_1),\ldots,(\emptyset;\Phi_n)\}\end{array}}{\Gamma_M,\Gamma_N;\Delta_M,\Delta_N;\mathcal{E}_M \sqcup \mathcal{E}_N \vdash MN\colon C} \multimap E_!$$

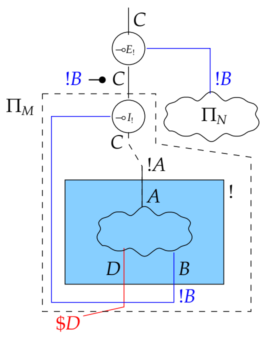


Figure 10: "Net" meaning of the rule $\multimap E_!$ in WALT

$$\begin{aligned}(\backslash x.M)N &\rightsquigarrow_w M \text{ if } \mathrm{no}(x,M)=0\\ (\backslash x.M)N &\rightsquigarrow_w M\{^N/_x\} \text{ if } \mathrm{no}(x,M)=1, \text{ and } N\in\Lambda_\forall\\ (\backslash x.M)N &\rightsquigarrow_w M\{^N/_x\} \text{ if } \mathrm{no}(x,M)>1, \text{ and } N\in\Lambda_\forall, \text{ and } \mathrm{FV}(N)\subseteq\{y\}\end{aligned}$$

$\rightsquigarrow_w^+$ is the transitive closure of $\rightsquigarrow_w$, while $\rightsquigarrow_w^*$ is the reflexive and transitive closure of $\rightsquigarrow_w$. M is in $\rightsquigarrow_w$-normal form, and we write $\mathrm{nf}(M)$, if $\rightsquigarrow_w$ cannot rewrite M anymore.

We conclude by recalling two main features of WALT:

Theorem 3.1 (Subject reduction ([Rov07]).) $\Gamma;\Delta;\mathcal{E}\vdash M{:}A$ *and* $M\rightsquigarrow_w^* N$, *imply* $\Gamma;\Delta;\mathcal{E}\vdash N{:}A$.

Theorem 3.2 (Polytime soundness ([Rov07]).) *Let* Π *be a derivation of WALT whose conclusion be* $\Gamma;\Delta;\mathcal{E}\vdash M{:}A$. *Let* $\mathrm{d}(\Pi)$ *be the maximal depth of* Π, *namely the maximal number of instances of the rules* $\$,!$ *that we can traverse moving from the conclusion of* Π, *to every of its axioms instances. Then, M normalizes to its normal form* $\mathrm{nf}(M)$ *in a number of steps, hence in a time, which is* $O(|\Pi|^{k^{\mathrm{d}(\Pi)}})$, *for some k.*

3.3 The combinators of WALT we need to recall

We recall the main aspects of combinators that can be typed in WALT, and which are required to show the completeness of WALT w.r.t. SRN. The details are in [Rov07].

(Binary) Words. They are the terms:

$$\begin{aligned}\overline{\overline{0}} &\equiv \backslash 01y.y\\ \overline{\overline{2^m+2^{m-1}\cdot v_{m-1}+\cdots+2^0\cdot v_0}} &\equiv \backslash 01y.v_0(\cdots(v_{m-1}(1\,y)\cdots)) \qquad (m\geq 1)\end{aligned}$$

that allow to encode the natural numbers in *binary* notation. Every word has type $\mathbf{W}\equiv\forall\alpha.!(\alpha\multimap\alpha)\multimap!(\alpha\multimap\alpha)\multimap\$(\alpha\multimap\alpha)$, where $m\geq 0$ and $v_{0\leq i\leq m-1}\in\{0,1\}$. Notice that every word is a Church numeral built using the two successors, identified by the variable names 0, and 1. The combinators `Ws0`, `Ws1`, and `P`, with type $\mathbf{W}\multimap\mathbf{W}$, and `B`, with type $\mathbf{W}\multimap\mathbf{W}\multimap\mathbf{W}\multimap\mathbf{W}$, exist. They are the two successors, the predecessor and the branching, respectively. The branching yields its second argument as result, if the first one is the word $\overline{\overline{0}}$. Otherwise, the result is the third argument.

Eager tensor. We need the eager tensor to represent tuples of λ-terms. For every $m \geq 1$, the type *eager tensor* is $\bigodot_{i=1}^{m} A_i \equiv \forall\alpha.(\mathbin{-\!\bullet}_{i=1}^{m} A_i \mathbin{-\!\bullet} \alpha) \multimap \alpha$. Its type constructors coincide to the standard tuples in the λ-calculus:

$$\langle\!\langle M_1 \dots M_m \rangle\!\rangle \equiv \backslash z.z\, M_1 \,\dots\, M_m \qquad (m \geq 1)$$
$$\backslash\langle\!\langle x_1 \dots x_m \rangle\!\rangle.M \equiv \backslash w.w(\backslash x_1 \dots x_m.M) \qquad (m \geq 1)$$

Here, we can fairly assume that only to closed terms can be used in $\langle\!\langle M_1 \dots M_m \rangle\!\rangle$. In [Rov07] the constraint is a little bit more weak. So, for every closed $M_1, \dots, M_m$, we have $(\backslash\langle\!\langle x_1 \cdots x_m \rangle\!\rangle.M)\ \langle\!\langle M_1, \dots, M_m \rangle\!\rangle \rightsquigarrow_w^{+} (\backslash x_1 \dots x_m.M)\, M_1 \dots M_m$.

Embedding. We can embed the arguments and the result of terms, with a functional type, into a suitable number of boxes. Since we have two kinds of implications, and we can transform the standard linear implication into an eager one, we have (at least) the following three kinds of embedding functors.

The *basic embedding* is $\texttt{Eb}^{n}[M] \equiv \backslash x.Mx$, for every $n \geq 1$. It takes a term M, with type $L \multimap \$^{m}A$, for any $m \geq 0$, and yields one of type $\$^{n}L \mathbin{-\!\bullet} \$^{m+n}A$.

The *linear embedding* is $\texttt{El}_{p}^{n}[M] \equiv \backslash x_1 \dots x_p.M\,x_1 \dots x_p$, for every $n, p \geq 0$. It takes a term M, with type $(\multimap_{i=1}^{p} L_i) \multimap \$^{m}A$, for every $m \geq 0$, and yields one of type $(\multimap_{i=1}^{p} \$^{n}L_i) \multimap \$^{m+n}A$.

The *eager embedding* $\texttt{Ee}_{p;q}^{n}[M]$ is:

$$\backslash w_1 \dots w_p z_1 \dots z_q.(\backslash w_1 \dots w_p.M w_1 \dots w_p z_1 \dots z_q)(\texttt{Eb}^{1}[\texttt{Coerce}^{n}]w_1) \dots (\texttt{Eb}^{1}[\texttt{Coerce}^{n}]w_p),$$

for every $n, p, q \geq 0$. It takes a term M of type $(\mathbin{-\!\bullet}_{i=1}^{p} \$\mathbf{W}) \mathbin{-\!\bullet} (\mathbin{-\!\bullet}_{j=1}^{q} \$^{m}L_j) \mathbin{-\!\bullet} \$^{m}A$ and yields one of type $(\mathbin{-\!\bullet}_{i=1}^{p} \$^{n}\mathbf{W}) \mathbin{-\!\bullet} (\mathbin{-\!\bullet}_{j=1}^{q} \$^{m+n}L_j) \mathbin{-\!\bullet} \$^{m+n}A$.

Coercing. The *coerce* function takes an instance of a binary word and reconstructs it inside a box. It is $\texttt{Coerce} \equiv \backslash n.(\backslash z.z\,\overline{\overline{0}})(n\,\texttt{Ws0}\,\texttt{Ws1})$. To our purposes, $\texttt{Coerce}$ must be iteratively composed to reconstruct a word into some given number of boxes:

$$\texttt{Coerce}^{0} \equiv \backslash x.x$$
$$\texttt{Coerce}^{1} \equiv \texttt{Coerce}$$
$$\texttt{Coerce}^{m+1} \equiv \backslash x.\texttt{El}_{1}^{1}[\texttt{Coerce}^{m}](\texttt{Coerce}^{1}\,x) \qquad (m \geq 1)$$

For every $m \geq 0$, $\texttt{Coerce}^{m}$ has type $\mathbf{W} \multimap \$^{m}\mathbf{W}$.

Eager diagonal. The *eager diagonal* ∇_{n}^{m} is:

$$\begin{aligned}\backslash w.(\backslash z.z\ \langle\!\langle \overbrace{\overline{\overline{0}}, \dots, \overline{\overline{0}}}^{n} \rangle\!\rangle)(w\ &(\backslash\langle\!\langle x_1 \dots x_n \rangle\!\rangle.\langle\!\langle \texttt{Eb}^{m}[\texttt{Ws0}]\,x_1, \dots, \texttt{Eb}^{m}[\texttt{Ws0}]\,x_n \rangle\!\rangle)\\ &(\backslash\langle\!\langle x_1 \dots x_n \rangle\!\rangle.\langle\!\langle \texttt{Eb}^{m}[\texttt{Ws1}]\,x_1, \dots, \texttt{Eb}^{m}[\texttt{Ws1}]\,x_n \rangle\!\rangle))\end{aligned}$$

for every $m, n \geq 1$. It combines the copies of the word, given as its input, by means of an elementary tensor constructor. Namely, $\nabla_{n}^{m}\,\overline{\overline{a}} \rightsquigarrow_w^{+} \langle\!\langle \overbrace{\overline{\overline{a}}, \dots, \overline{\overline{a}}}^{n} \rangle\!\rangle$. Every copy is generated from scratch, by iterating the successors on words. The result is contained into a single box, but every component of the elementary tensor, in the result, is m boxes deep. Namely, ∇_{n}^{m} has type $\mathbf{W} \multimap \$(\bigodot_{i=1}^{n} \$^{m}\mathbf{W})$.

Iterator. For every $\mathsf{n}, \mathsf{s} \geq 0$, and $m \geq 1$, and for every closed term G_0, G_1, and G_2, all with type $\$\mathbf{W} \multimap (\multimap_{i=1}^{\mathsf{n}} \$\mathbf{W}) \multimap (\multimap_{j=1}^{\mathsf{s}} \$^m\mathbf{W}) \multimap \$^m\mathbf{W} \multimap \$^m\mathbf{W}$, the iterator $\mathtt{It}_{1+\mathsf{n};\mathsf{s}}[G_0, G_1, G_2]$ has type:

$$\$\mathbf{W} \multimap (\multimap_{i=1}^{\mathsf{n}} \$\mathbf{W}) \multimap (\multimap_{i=1}^{\mathsf{s}} \$^{m+4}\mathbf{W}) \multimap \$^{m+4}\mathbf{W}$$

As expected, the first argument of $\mathtt{It}_{1+\mathsf{n};\mathsf{s}}[G_0, G_1, G_2]$ drives the iteration, G_2 is the base function, while G_0 and G_1 are the inductive ones, chosen by the "bits" in the first argument itself of $\mathtt{It}_{1+\mathsf{n};\mathsf{s}}[G_0, G_1, G_2]$.

To formally recall the behavior of the iterator, we need to assume that: (i) $\mathsf{n}, \mathsf{s} \geq 0, \overline{\overline{a}}, \overline{\overline{n}}, \overline{\overline{n_1}}, \ldots, \overline{\overline{n_{\mathsf{n}}}}, \overline{\overline{s_1}}, \ldots, \overline{\overline{s_{\mathsf{s}}}}$ be some words, (ii) $\{v_0, v_1, \ldots\}$ be a denumerable set of metavariables to range over $\{0, 1\}$, (iii) $[\overline{\overline{x}}]^i$ denotes a list with $i \in \mathbb{N}$ copies of the word $\overline{\overline{x}}$, for any x.

Also, we assume that:

- $G_2\, \overline{\overline{0}}\, \overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\, \overline{\overline{s_1}} \ldots \overline{\overline{s_{\mathsf{s}}}}\, \overline{\overline{0}}$ rewrites to a word $\overline{\overline{a}}$, and
- $G_1\, \overline{\overline{0}}\, \overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\, \overline{\overline{s_1}} \ldots \overline{\overline{s_{\mathsf{s}}}}\, \overline{\overline{a}}$ rewrites to a word, denoted as $r[0, a, n_1, \ldots, n_{\mathsf{n}}, s_1, \ldots, s_{\mathsf{s}}]$, and
- for every m, i, such that $m \geq 0, m - 1 \geq i \geq 0$,

$$G_{v_i} \overline{\overline{\left(\sum_{j=0}^{m-(i+1)} 2^{m-(i+1)-j} v_{m-j} \right)}}\, \overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\, \overline{\overline{s_1}} \ldots \overline{\overline{s_{\mathsf{s}}}}\, r[m-(i+1), a, n_1, \ldots, n_{\mathsf{n}}, s_1, \ldots, s_{\mathsf{s}}]$$

  rewrites to a word, denoted as $r[m - i, a, n_1, \ldots, n_{\mathsf{n}}, s_1, \ldots, s_{\mathsf{s}}]$.

Then:

$$\mathtt{It}_{1+\mathsf{n};\mathsf{s}}[G_0, G_1, G_2]\, \overline{\overline{0}}\, \overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\, \overline{\overline{s_1}} \ldots \overline{\overline{s_{\mathsf{s}}}} \rightsquigarrow_w^+ \overline{\overline{a}}$$

$$\mathtt{It}_{1+\mathsf{n};\mathsf{s}}[G_0, G_1, G_2] \overline{\overline{\left(\sum_{j=0}^{m} 2^j v_j \right)}}\, \overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\, \overline{\overline{s_1}} \ldots \overline{\overline{s_{\mathsf{s}}}} \rightsquigarrow_w^+ r[m, a, n_1, \ldots, n_{\mathsf{n}}, s_1, \ldots, s_{\mathsf{s}}] \qquad \left(\overline{\overline{\sum_{j=0}^{m} 2^j v_j}} \neq 0 \right)$$

The full details about $\mathtt{It}_{1+\mathsf{n};\mathsf{s}}[G_0, G_1, G_2]$ are in [Rov07], whose keypoint is to prove that such a combinator is indeed representable inside WALT, giving its completeness w.r.t. CISRN.

4 Safe Recursion on Notation

We recall two classes of functions: *Safe recursion on notation* (SRN) [BC92], and *Composition-linear safe recursion on notation* (ClSRN), both in the style of [BW96]. Remark that Composition-linear safe recursion on notation was identified as *Quasi-linear safe recursion on notation* in [Rov07]. The reason for the name changing will be given in the conclusions (Section 7).

The signature of Safe recursion on notation. Let $\Sigma_{\mathsf{SRN}} = \cup_{k,l\in\mathbb{N}}\Sigma^{k,l}_{\mathsf{SRN}}$ be the signature of Safe recursion on notation . Σ_{SRN} contains the *base functions* and it is closed under the schemes called *safe composition* and *safe recursion*. For every $k, l \in \mathbb{N}$, the *base functions* are the *zero* $\mathsf{z}^{k;l} \in \Sigma^{k;l}_{\mathsf{SRN}}$, the *successors* $\mathsf{s}^{0;1}_0, \mathsf{s}^{0;1}_1$, the *predecessor* $\mathsf{p}^{0;1} \in \Sigma^{0;1}_{\mathsf{SRN}}$, the *projection* $\pi^{k;l}_i \in \Sigma^{k;l}_{\mathsf{SRN}}$, with $1 \le i \le k + l$, and the *branching* $\mathsf{c}^{0,3} \in \Sigma^{0;3}_{\mathsf{SRN}}$.

The *safe composition* is $\circ^{k;l}_{k';l'}[f, g_1, \dots, g_{k'}, h_1, \dots, h_{l'}] \in \Sigma^{k;l}_{\mathsf{SRN}}$ if $f \in \Sigma^{k';l'}_{\mathsf{SRN}}$, $g_1, \dots, g_{k'} \in \Sigma^{k;0}_{\mathsf{SRN}}$, and $h_i \in \Sigma^{k;l}_{\mathsf{SRN}}$, for every $k, k', l, l' \in \mathbb{N}$.

The *safe recursion* is $\mathsf{r}^{k+1;l}[g, h_0, h_1] \in \Sigma^{k+1;l}_{\mathsf{SRN}}$ if $g \in \mathsf{SRN}^{k;l}$, and $h_0, h_1 \in \Sigma^{k+1;l+1}_{\mathsf{SRN}}$.

Safe recursion on notation. Let V be a denumerable set of *names of variables*, disjoint from Σ_{SRN}. The set Safe recursion on notation (SRN) contains functions with signature Σ_{SRN}. SRN is defined as follows. $\mathsf{V} \subset \mathsf{SRN}$, and for every $k, l \in \mathbb{N}$, if $f \in \Sigma^{k;l}_{\mathsf{SRN}}$, and $t_1, \dots, t_k, u_1, \dots, u_l \in \mathsf{SRN}$, then $f(t_1, \dots, t_k, u_1, \dots, u_l) \in \mathsf{SRN}$. A term is *closed* if it does not contain variables of V.

Notations and terminology. $x, y, z \dots$ denote elements of V. $t, u, v \dots$ denote elements of SRN. For every $f \in \Sigma^{k,l}_{\mathsf{SRN}}$, k and l are the *normal* and *safe* arity of f, respectively. For every $k, l \in \mathbb{N}$, such that $l - k \ge 1$, $\vec{t}_{[k;l]}$ denotes a *non empty* sequence $t_k, \dots, t_l$ of $l - k + 1$ terms in SRN. $\vec{t}_{[k;l]}(i)$, with $k \le i \le l$, denotes the element t_i of $\vec{t}_{[k;l]}$.

An equational theory on SRN. The definition of the equational theory exploits that every natural number n can be written, uniquely, in binary notation, as $\sum^{m}_{j=0} 2^{m-j} v_{m-j}$. So, assuming to abbreviate the base functions $\mathsf{s}^{0;1}_0, \mathsf{s}^{0;1}_1$ as $\mathsf{s}_0, \mathsf{s}_1$, respectively, we can follow [MO04] and say that 0 is equivalent to $\mathsf{z}^{0;0}$, and $n \ge 1$ to $\mathsf{s}_{v_0}(\dots(\mathsf{s}_{v_{m-1}}(\mathsf{s}_1\, \mathsf{z}^{0;0}))\dots)$. Then, the *equational theory* is as follows. *Zero* is constantly equal to 0: $\mathsf{z}^{k;l}(\vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l]}) = 0$ for any $k, l \in \mathbb{N}$. The *predecessor* erases the least significant bit of any number greater than 0: for every $i \in \{0, 1\}$, $\mathsf{p}^{0;1}(0) = 0$, and $\mathsf{p}^{0;1}(\mathsf{s}_i(y)) = y$. We shall use p as an abbreviation of $\mathsf{p}^{0;1}$. The *conditional* has three arguments. If the first is zero, then the result is the second argument. Otherwise, it is the third one: for every $i \in \{0, 1\}$, $\mathsf{c}^{0,3}(0, y_0, y_1) = y_0$, and $\mathsf{c}^{0,3}(\mathsf{s}_i(y), y_0, y_1) = y_1$. The *projection* chooses one argument, out of a given tuple, as a result: for every $1 \le i \le k + l$, $\pi^{k;l}_i(\vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l]}) = x_i$.

The *safe composition* uses as arguments of f both the results of the *normal functions* $g_1, \dots, g_{k'}$, applied to k normal arguments, and the result of the *safe functions* $h_1, \dots, h_{l'}$, applied to k normal and l safe arguments:

$$\begin{aligned}&\circ^{k;l}_{k';l'}\,[f, g_1, \dots, g_{k'}, h_1, \dots, h_{l'}](\vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l]})\\ &\qquad = f\,(g_1(\vec{x}_{[1;k]}), \dots, g_{k'}(\vec{x}_{[1;k]}), h_1(\vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l]}), \dots, h_{l'}(\vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l]}))\ .\end{aligned}$$

The *recursion* iterates either the function h_0, or h_1, as many times as the length of its first argument. The choice between h_0, and h_1 depends on the least significant digit of the first argument, while the base of the iteration is a function g. The recursion is:

$$\begin{aligned}\mathsf{r}^{k+1;l}[g, h_0, h_1](0, \vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l]}) &= g(\vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l]})\\ \mathsf{r}^{k+1;l}[g, h_0, h_1](\mathsf{s}_i(x), \vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l]}) &= h_i(x, \vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l]}, \mathsf{r}^{k+1;l}[g, h_0, h_1](x, \vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l]}))\ .\end{aligned}$$

4.1 Composition-linear safe recursion on notation

ClSRN is SRN with a restricted form of safe composition, which we call *linear (safe) composition*. Its signature is $\diamond_{k';l'}^{k;\sum_{i=1}^{l'} l_i}[f, g_1, \ldots, g_{k'}, h_1, \ldots, h_{l'}] \in \Sigma_{\mathsf{ClSRN}}^{k;\sum_{i=1}^{l'} l_i}$ if $f \in \Sigma_{\mathsf{ClSRN}}^{k';l'}$, $g_1, \ldots, g_{k'} \in \Sigma_{\mathsf{ClSRN}}^{k;0}$, and $h_i \in \Sigma_{\mathsf{ClSRN}}^{k;l_i}$, with $i \in \{1, \ldots, l'\}$, for every $k, l, l', l_1, \ldots, l_{l'} \in \mathbb{N}$. Namely, unlike the general safe composition scheme of SRN, the safe arguments are used linearly: the list of safe arguments is split into as many sub-sequences as required by the safe arity of every safe function h_j, with $1 \leq j \leq l'$:

$$\begin{aligned}
&\diamond_{k';l'}^{k;\sum_{i=1}^{l'} l_i}[f, g_1, \ldots, g_{k'}, h_1, \ldots, h_{l'}]\,(\vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l_1]}, \vec{x}_{[k+1+l_1;k+l_1+l_2]}, \ldots, \vec{x}_{[k+1+\sum_{i=1}^{l'-1} l_i;k+1+\sum_{i=1}^{l'} l_i]}) \\
&\qquad = f\,(g_1(\vec{x}_{[1;k]}), \ldots, g_{k'}(\vec{x}_{[1;k]}), h_1(\vec{x}_{[1;k]}, \vec{x}_{[k+1;k+l_1]}), \ldots, h_{l'}(\vec{x}_{[1;k]}, \vec{x}_{[k+1+\sum_{i=1}^{l'-1} l_i;k+1+\sum_{i=1}^{l'} l_i]}))\ .
\end{aligned}$$

5 The full safe composition of SRN in WALT

We know that WALT contains ClSRN as its subsystem. Namely, it holds:

Theorem 5.1 ([Rov07].) *There is a map* $[\![\ ]\!]$*, such that, if* $f(n_1 \ldots, n_k, s_1 \ldots, s_l)$ *belongs to* ClSRN*, and* $f(n_1, \ldots, n_k, s_1, \ldots, s_l) = n$*, then* $[\![f(n_1, \ldots, n_k, s_1, \ldots, s_l)]\!] \rightsquigarrow^*_w \overline{\overline{n}}$*, for every* $n_1, \ldots, n_k, s_1, \ldots, s_l, n \in \mathbb{N}$*, in binary notation.*

Here we go further by defining the combinators that, using the base combinators of Section 3.3, give the *full* safe composition scheme of SRN as a term of WALT. The definitions here below will realize the functional blocks of the example in Figure 5.

Sharing safe names. Let $\mathsf{n}, \mathsf{s} \geq 0$, $m \geq 1$, and a closed term M, with type $(\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{s}+1}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}$, be given. We call $\mathtt{Y}^{\mathsf{n};\mathsf{s}}_m$ the closed term that takes M and $\mathsf{n} + \mathsf{s}$ words as its arguments. The first n arguments can be viewed as a normal ones, while the s second ones as safe. Then, $\mathtt{Y}^{\mathsf{n};\mathsf{s}}_m$ applies M to the normal and safe arguments, using $\overline{\overline{s_{\mathsf{s}}}}$ as a value for the two last safe positions. Namely, the *behavior* is:

$$\mathtt{Y}^{\mathsf{n};\mathsf{s}}_m[M]\overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\,\overline{\overline{s_1}} \ldots \overline{\overline{s_{\mathsf{s}}}} \rightsquigarrow^+_w M\,\overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\,\overline{\overline{s_1}} \ldots \overline{\overline{s_{\mathsf{s}}}}\,\overline{\overline{s_{\mathsf{s}}}}$$

Clearly, using the same safe value twice, after its duplication, we are sharing it.

The *type* of $\mathtt{Y}^{\mathsf{n};\mathsf{s}}_m$ is:

$$((\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{s}+1}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}) \multimap (\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{s}}_{j=1} \$^{m+4}\mathbf{W}) \multimap \$^{m+4}\mathbf{W}$$

The *definition* of $\mathtt{Y}^{\mathsf{n};\mathsf{s}}_m$ is:

$$\mathtt{It}_{1+\mathsf{n};\mathsf{s}}[\backslash w\, x_1 \ldots x_{\mathsf{n}}\, y_1 \ldots y_{\mathsf{s}+1}.\overline{\overline{0}}, \backslash w.M, \backslash w\, x_1 \ldots x_{\mathsf{n}}\, y_1 \ldots y_{\mathsf{s}+1}.y_{\mathsf{s}}]\,\overline{\overline{1}}$$

Rotating safe names. Let $\mathsf{n}, \mathsf{s} \geq 0$, $m \geq 1$, and a closed term M, with type $(\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{s}}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}$, be given. We call $\circlearrowleft^{\mathsf{n};\mathsf{s}}_m$ the closed term that takes M and $\mathsf{n} + \mathsf{s}$ words as its arguments. The first n arguments can be viewed as a normal ones, while the second s as safe. $\circlearrowleft^{\mathsf{n};\mathsf{s}}_m$ applies M to the normal arguments in the given order, while using $\overline{\overline{s_1}}$ as value at position s, shifting all the others leftward. Namely, the *behavior* is:

$$\circlearrowleft^{\mathsf{n};\mathsf{s}}_m [M]\overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\,\overline{\overline{s_1}} \ldots \overline{\overline{s_{\mathsf{s}}}} \rightsquigarrow^*_w M\,\overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\,\overline{\overline{s_2}} \ldots \overline{\overline{s_{\mathsf{s}}}}\,\overline{\overline{s_1}}$$

The *type* of $\circlearrowleft^{\mathsf{n};\mathsf{s}}_m$ is:

$$((\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{s}}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}) \multimap (\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{s}}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}$$

The *definition* of $\circlearrowleft^{\mathsf{n};\mathsf{s}}_m$ is $\backslash x_1 \ldots x_{\mathsf{n}}\, y_{\mathsf{s}}\, y_1 \ldots y_{\mathsf{s}-1}.M\, x_1 \ldots x_{\mathsf{n}}\, y_1 \ldots y_{\mathsf{s}}$.

Multiple sharing of safe names. Let $\mathsf{n}, \mathsf{p}, \mathsf{q} \geq 0$, and $m \geq 1$. Let M be a closed term with type $(\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{p}+\mathsf{q}}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}$, when $\mathsf{p} \geq 1$, and $(\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap \$^m\mathbf{W}$, when $\mathsf{p} = 0$. We call $\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m$ the closed term that takes M and $\mathsf{n} + \mathsf{p} + \mathsf{q}$ words as its arguments. The first n arguments can be viewed as normal ones, while the second $\mathsf{p} + \mathsf{q}$ as safe. If $\mathsf{p} > 0$, and $\mathsf{q} \geq 1$, then $\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m$ applies M to the normal and safe arguments, using $\overline{\overline{s_{\mathsf{p}}}}$ as a value for the last q safe positions. Namely, the *behavior* is:

$$\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m[M]\,\overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\,\overline{\overline{s_1}} \ldots \overline{\overline{s_{\mathsf{p}}}} \rightsquigarrow^*_w M\,\overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\,\overline{\overline{s_1}} \ldots \overline{\overline{s_{\mathsf{p}}}}\,\overbrace{\overline{\overline{s_{\mathsf{p}}}} \ldots \overline{\overline{s_{\mathsf{p}}}}}^{\mathsf{q}} \qquad (\mathsf{p} > 0, \mathsf{q} \geq 1)$$

Otherwise, $\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m[M]$ coincides to M.

The *type* of $\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m$ is:

$$((\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{p}+\mathsf{q}}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}) \multimap (\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{p}}_{j=1} \$^{m+4\mathsf{q}}\mathbf{W}) \multimap \$^{m+4\mathsf{q}}\mathbf{W} \qquad (\mathsf{p} \geq 1)$$

$$((\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap \$^m\mathbf{W}) \multimap (\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap \$^m\mathbf{W} \qquad (\mathsf{p} = 0)$$

The *definition* of $\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m$ is:

$$\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m[M] \equiv M \qquad (\mathsf{p} = 0 \text{ or } \mathsf{q} = 0)$$

$$\mathtt{mY}^{\mathsf{n};(\mathsf{p},1)}_m[M] \equiv \mathtt{Y}^{\mathsf{n};\mathsf{p}}_m[M] \qquad (\mathsf{p} > 0)$$

$$\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m[M] \equiv \mathtt{Y}^{\mathsf{n};\mathsf{p}}_{m+4(\mathsf{q}-1)}[\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q}-1)}_m[M]] \qquad (\mathsf{p} > 0, \mathsf{q} > 1)$$

Multiple sharing and rotation of safe names. Let $\mathsf{n}, \mathsf{p}, \mathsf{q} \geq 0$, and $m \geq 1$. Let M be a closed term with type $(\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{p}+\mathsf{q}}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}$, when $\mathsf{p} \geq 1$, and $(\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap \$^m\mathbf{W}$, when $\mathsf{p} = 0$. We call $\circlearrowleft\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m$ the closed term that takes M and $\mathsf{n} + \mathsf{p} + \mathsf{q}$ words as its arguments. The first n arguments can be viewed as normal ones, while the last $\mathsf{p} + \mathsf{q}$ as safe ones. If $\mathsf{p} \geq 1$, then $\circlearrowleft\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m$ applies M to the normal arguments in the given order, while using $\overline{\overline{s_1}}$ as value in the last q positions, shifting all the others leftward. Namely, the *behavior* is:

$$\circlearrowleft\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m[M]\,\overline{\overline{n_1}}\ldots\overline{\overline{n_{\mathsf{n}}}}\,\overline{\overline{s_1}}\,\overline{\overline{s_2}}\ldots\overline{\overline{s_{\mathsf{p}}}} \rightsquigarrow^*_w M\,\overline{\overline{n_1}}\ldots\overline{\overline{n_{\mathsf{n}}}}\,\overline{\overline{s_2}}\ldots\overline{\overline{s_{\mathsf{p}}}}\,\overbrace{\overline{\overline{s_1}}\ldots\overline{\overline{s_1}}}^{\mathsf{q}} \qquad (\mathsf{p} \geq 1)$$

Otherwise, with $\mathsf{p} = 0$, $\circlearrowleft\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m[M]$ coincides to M, for any q.

The *type* of $\circlearrowleft\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m$ is:

$$((\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{p}+\mathsf{q}}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}) \multimap (\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{p}}_{j=1} \$^{m+4\mathsf{q}}\mathbf{W}) \multimap \$^{m+4\mathsf{q}}\mathbf{W} \qquad (\mathsf{p} \geq 1)$$

$$((\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap \$^m\mathbf{W}) \multimap (\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap \$^m\mathbf{W} \qquad (\mathsf{p} = 0)$$

The *definition* of $\circlearrowleft\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m$ is:

$$\circlearrowleft\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m[M] \equiv M \qquad (\mathsf{p} = 0 \text{ or } \mathsf{q} = 0)$$

$$\circlearrowleft\mathtt{mY}^{\mathsf{n};(1,\mathsf{q})}_m[M] \equiv \mathtt{mY}^{\mathsf{n};(1,\mathsf{q})}_m[M] \qquad (\mathsf{p} = 1)$$

$$\circlearrowleft\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m[M] \equiv \circlearrowleft^{\mathsf{n};\mathsf{p}}_{m+4\mathsf{q}}[\mathtt{mY}^{\mathsf{n};(\mathsf{p},\mathsf{q})}_m[M]] \qquad (\mathsf{p} > 1)$$

Square composition. The *intuitive side* first. Let $G_1, \ldots, G_{\mathsf{n}'}$ be terms that we call *normal* for we think of them as functions with only normal arity n. Analogously, let $H_1, \ldots, H_{\mathsf{s}'}$ be terms that we call *safe* since we look at them as functions with normal arity n, and safe arity s_j, for every $1 \leq j \leq \mathsf{s}'$. Let $\mathsf{s} = \max\{\mathsf{s}_1, \ldots, \mathsf{s}_{\mathsf{s}'}, \mathsf{s}'\}$; notice that s is determined comparing the safe arguments of every safe term and their total number s'. The behavior of $\boxdot^{\mathsf{n};\mathsf{s}}_{\mathsf{n}'}[F, G_1 \ldots G_{\mathsf{n}'}, H_1 \ldots H_{\mathsf{s}'}]$ comprises some phases, of which we have an example of result, contained in the innermost dashed box, labeled $\boxdot^{1;3}_1$, of Figure 5. Every normal argument of $\boxdot^{\mathsf{n};\mathsf{s}}_{\mathsf{n}'}$ is replicated as many times as $\mathsf{n}' + \mathsf{s}$ so that every copy can be dispatched to normal and safe terms.

Then, the term F is used to generate F' with n' normal and s safe arities. F' behaves like F once erased its $\mathsf{s} - \mathsf{s}'$ arguments. For example, the bullet aside $[f]^\circ$, which plays the role of F in Figure 5, represents the extension of F', with respect to F, that erases its third safe argument. The generation of H'_j from H_j, with $1 \leq j \leq \mathsf{s}_j$, is analogous to the one of F', from F: if necessary, every H'_j erases $\mathsf{s} - \mathsf{s}_j$ safe arguments. If F' has to erase $\mathsf{s}-\mathsf{s}'$ safe arguments, we supply them as the result of $\mathsf{s}-\mathsf{s}'$ fake functions that erase all of their arguments and give a word as result. In Figure 5 there is a single fake function named $\overline{\overline{0}}\bullet\bullet\bullet$, yielding $\overline{\overline{0}}$.

The normal and safe arguments of $\boxdot^{\mathsf{n};\mathsf{s}}_{\mathsf{n}'}$ are replicated by using two different processes. The one for normal arguments is the standard eager diagonal, building every copy from scratch. Instead, every replica of a safe argument is obtained by using the above combinator that rotates and shares multiple safe values. Once all the required copies of safe arguments are at hand, they are rearranged, and appropriately distributed to $H'_1,\ldots,H'_{\mathsf{s}}$.

Now, the *technical side*. Let us assume to have a set of closed terms $F, G_1,\ldots,G_{\mathsf{n}'},H_1,\ldots,H_{\mathsf{s}'}$ with the following types, respectively:

$$\begin{array}{ll}
(\multimap^{\mathsf{n}'}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{s}'}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W} & \\
(\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap \$^m\mathbf{W} & (i\in\{1,\ldots,\mathsf{n}'\}) \\
(\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{s}_j}_{k=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W} & (j\in\{1,\ldots,\mathsf{s}'\})
\end{array}$$

Let $\mathsf{s}=\max\{\mathsf{s}_1,\ldots,\mathsf{s}_{\mathsf{s}'},\mathsf{s}'\}$.

The *type* of $\boxdot^{\mathsf{n};\mathsf{s}}_{\mathsf{n}'}$ is:

$$\begin{array}{l}
((\multimap^{\mathsf{n}'}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{s}'}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}) \multimap \\
\quad (\multimap^{\mathsf{n}'}_{k=1} ((\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap \$^m\mathbf{W})) \multimap \\
\quad\quad (\multimap^{\mathsf{s}'}_{j=1} ((\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{s}_j}_{k=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W})) \multimap \\
\quad\quad\quad (\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{s}^2}_{k=1} \$^{2m+1}\mathbf{W}) \multimap \$^{2m+1}\mathbf{W}
\end{array}$$

The *definition* of $\boxdot^{\mathsf{n};\mathsf{s}}_{\mathsf{n}'}$ is:

$$\boxdot^{\mathsf{n};\mathsf{s}}_{\mathsf{n}'}[F,G_1,\ldots,G_{\mathsf{n}'},H_1,\ldots,H_{\mathsf{s}'}] \equiv \backslash n_1\ldots n_{\mathsf{n}}.\mathtt{Ee}^2_{0;\mathsf{n}+\mathsf{s}}[G](\mathtt{El}^1_1[\nabla^1_{\mathsf{n}'+\mathsf{s}}]\,n_1)\ldots(\mathtt{El}^1_1[\nabla^1_{\mathsf{n}'+\mathsf{s}}]\,n_{\mathsf{n}})$$

where:

$$\begin{array}{rll}
G \equiv & \backslash\langle\!\langle x_{11}\ldots x_{\mathsf{n}'1}y_{11}\ldots y_{\mathsf{s}1}\rangle\!\rangle\ldots\backslash\langle\!\langle x_{1\mathsf{n}}\ldots x_{\mathsf{n}'\mathsf{n}}y_{1\mathsf{n}}\ldots y_{\mathsf{s}\mathsf{n}}\rangle\!\rangle. & \\
& \quad\backslash w_{11}w_{12}\ldots w_{1\mathsf{s}}.\backslash w_{21}w_{22}\ldots w_{2\mathsf{s}}.\ldots\ldots.\backslash w_{\mathsf{s}1}w_{\mathsf{s}2}\ldots w_{\mathsf{s}\mathsf{s}}. & \\
& \quad\mathtt{Ee}^{m-1}_{0;\mathsf{n}'+\mathsf{s}}[F']\,(G_1\,x_{11}\ldots x_{1\mathsf{n}})\ldots(G_{\mathsf{n}'}\,x_{\mathsf{n}'1}\ldots x_{\mathsf{n}'\mathsf{n}}) & \\
& \quad(\mathtt{Ee}^{m-1}_{0;\mathsf{n}+\mathsf{s}}[H'_1]\,(\mathtt{El}^1_1[\mathtt{Coerce}^{m-1}]\,y_{11})\ldots(\mathtt{El}^1_1[\mathtt{Coerce}^{m-1}]\,y_{1\mathsf{n}})\,w_{11}\,w_{21}\ldots w_{\mathsf{s}1}) & \\
& \quad(\mathtt{Ee}^{m-1}_{0;\mathsf{n}+\mathsf{s}}[H'_2]\,(\mathtt{El}^1_1[\mathtt{Coerce}^{m-1}]\,y_{11})\ldots(\mathtt{El}^1_1[\mathtt{Coerce}^{m-1}]\,y_{1\mathsf{n}})\,w_{12}\,w_{22}\ldots w_{\mathsf{s}2})\ \ldots & \\
& \quad\ldots(\mathtt{Ee}^{m-1}_{0;\mathsf{n}+\mathsf{s}}[H'_{\mathsf{s}}]\,(\mathtt{El}^1_1[\mathtt{Coerce}^{m-1}]\,y_{\mathsf{s}1})\ldots(\mathtt{El}^1_1[\mathtt{Coerce}^{m-1}]\,y_{\mathsf{s}\mathsf{n}})\,w_{1\mathsf{s}}\,w_{2\mathsf{s}}\ldots w_{\mathsf{s}\mathsf{s}}) & \\
F' \equiv & \backslash x_1\ldots x_{\mathsf{n}'}y_1\ldots y_{\mathsf{s}}.F\,x_1\ldots x_{\mathsf{n}'}y_1\ldots y_{\mathsf{s}'} & \\
H'_i \equiv & \backslash z_{i1}\ldots z_{i\mathsf{n}}w_{i1}\ldots w_{i\mathsf{s}_i}w_{i\mathsf{s}_i+1}\ldots w_{i\mathsf{s}}.H_i\,z_{i1}\ldots z_{i\mathsf{n}}\,w_{i1}\ldots w_{i\mathsf{s}_i} & (i\in\{1,\ldots,\mathsf{s}'\}) \\
H'_j \equiv & \backslash z_{j1}\ldots z_{j\mathsf{n}}w_{j1}\ldots w_{j\mathsf{s}}.\overline{\overline{0}} & (j\in\{\mathsf{s}',\ldots,\mathsf{s}-\mathsf{s}'\})
\end{array}$$

G takes both n copies of the $\mathsf{n}'+\mathsf{s}$ normal arguments, generated by the n instances of $\mathtt{El}^1_1[\nabla^1_{\mathsf{n}'+\mathsf{s}}]$, and s^2 safe arguments. Then, it dispatches them to the terms $G_1,\ldots,G_{\mathsf{n}'}$ and $H'_1,\ldots,H'_{\mathsf{s}}$. As we said, every H'_i takes s safe arguments and supplies only the first s_i to H_i.

The *behavior* is:

$$\boxdot^{\mathsf{n};\mathsf{s}}_m[F,G_1\ldots G_{\mathsf{n}'},H_1\ldots H_{\mathsf{s}'}]\,\overline{\overline{n}}_1\ldots\overline{\overline{n}}_{\mathsf{n}}\,\overbrace{\overline{\overline{s}}_1\ldots\overline{\overline{s}}_1}^{\mathsf{s}}\ldots\ldots\overbrace{\overline{\overline{s}}_{\mathsf{s}}\ldots\overline{\overline{s}}_{\mathsf{s}}}^{\mathsf{s}} \rightsquigarrow^+_w F\,\overline{\overline{g_1}}\ldots\overline{\overline{g_{\mathsf{n}'}}}\,\overline{\overline{h_1}}\ldots\overline{\overline{h_{\mathsf{s}'}}}$$

if $G_i\,\overline{\overline{n}}_1\ldots\overline{\overline{n}}_{\mathsf{n}} \rightsquigarrow^*_w \overline{\overline{g_i}}$, with $1\leq i\leq \mathsf{n}'$, and $H_j\,\overline{\overline{n}}_1\ldots\overline{\overline{n}}_{\mathsf{n}}\,\overline{\overline{s}}_1\ldots\overline{\overline{s}}_{\mathsf{s}} \rightsquigarrow^*_w \overline{\overline{h_j}}$, with $1\leq j\leq \mathsf{s}'$.

5.1 Multiple sharing and rotation of safe names in a square composition.

Let $\mathsf{n}, \mathsf{p}, i \geq 0$, $m \geq 1$, and a closed term M with type $(\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{p}^2}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}$ be given. We call $\curlyvee^{\mathsf{n};\mathsf{p}\backslash i}_m$ the closed term that takes M and $\mathsf{n} + i + \mathsf{p}\, i$ words as its arguments. The first n arguments can be viewed as normal ones, while the last $i + \mathsf{p}\, i$ as safe ones. Then, $\curlyvee^{\mathsf{n};\mathsf{p}\backslash i}_m[M]$ replicates every of the i safe arguments $\overline{\overline{s_{\mathsf{p}-i+1}}} \ldots \overline{\overline{s_{\mathsf{p}}}}$ as many times as p. Finally the "blocks" $\overline{\overline{s_1}} \ldots \overline{\overline{s_1}} \ldots\ldots \overline{\overline{s_{\mathsf{p}-i}}} \ldots \overline{\overline{s_{\mathsf{p}-i}}}\, \overline{\overline{s_{\mathsf{p}-i+1}}} \ldots \overline{\overline{s_{\mathsf{p}-i+1}}} \ldots\ldots \overline{\overline{s_{\mathsf{p}}}} \ldots \overline{\overline{s_{\mathsf{p}}}}$, with p elements each, are used as the p^2 safe arguments of M. Namely, the *behavior* is:

$$\curlyvee^{\mathsf{n};\mathsf{p}\backslash i}_m[M]\, \overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\, \overbrace{\overline{\overline{s_{\mathsf{p}-i+1}}} \ldots \overline{\overline{s_{\mathsf{p}}}}}^{i}\, \overbrace{\overline{\overline{s_1}} \ldots \overline{\overline{s_1}}}^{\mathsf{p}} \ldots\ldots \overbrace{\overline{\overline{s_i}} \ldots \overline{\overline{s_i}}}^{\mathsf{p}} \rightsquigarrow^*_w M\, \overline{\overline{n_1}} \ldots \overline{\overline{n_{\mathsf{n}}}}\, \overbrace{\overline{\overline{s_1}} \ldots \overline{\overline{s_1}}}^{\mathsf{p}} \ldots\ldots \overbrace{\overline{\overline{s_{\mathsf{p}}}} \ldots \overline{\overline{s_{\mathsf{p}}}}}^{\mathsf{p}}$$

The *type* of $\curlyvee^{\mathsf{n};\mathsf{p}\backslash i}_m[M]$ is:

$$((\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{p}^2}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}) \multimap (\multimap^{\mathsf{n}}_{j=1} \$\mathbf{W}) \multimap (\multimap^{i+\sum_{k=1}^{\mathsf{p}-i} \mathsf{p}}_{j=1} \$^{m+4(\mathsf{p}-1)i}\mathbf{W}) \multimap \$^{m+4(\mathsf{p}-1)i}\mathbf{W}$$

The *definition* of $\curlyvee^{\mathsf{n};\mathsf{p}\backslash i}_m[M]$ is:

$$\begin{aligned}
\curlyvee^{\mathsf{n};\mathsf{p}\backslash i}_m[M] &\equiv M && (\mathsf{p} \leq 1 \text{ or } i = 0)\\
\curlyvee^{\mathsf{n};\mathsf{p}\backslash 1}_m[M] &\equiv \circlearrowleft\!\mathtt{mY}^{\mathsf{n};(1+\sum_{k=1}^{\mathsf{p}-1} \mathsf{p},\mathsf{p}-1)}_m[M] &&\\
\curlyvee^{\mathsf{n};\mathsf{p}\backslash i}_m[M] &\equiv \circlearrowleft\!\mathtt{mY}^{\mathsf{n};(i-1+\sum_{k=1}^{\mathsf{p}-(i-1)} \mathsf{p},\mathsf{p})}_{m+4(\mathsf{p}-1)(i-1)}[\curlyvee^{\mathsf{n};\mathsf{p}\backslash i-1}_m[M]] && (\mathsf{p} \geq i > 0)
\end{aligned}$$

6 SRN-completeness of WALT

We extend the completeness of WALT from ClSRN [Rov07] to SRN. The key ingredients are square composition and the multiple sharing of safe names of Section 5.

Functions of SRN into WALT. We start defining a map $[\ ]^\circ$ from the signature Σ_{SRN} to terms of WALT. Its clauses are identical to those mapping ClSRN to WALT [Rov07], but the one mapping the composition. of course. Here they are:

1. $[\mathtt{z}^{0;0}]^\circ \equiv \mathtt{El}^1_0[\overline{\overline{0}}]$, while $[\mathtt{z}^{k;l}]^\circ \equiv \backslash n_1 \ldots n_k\, s_1 \ldots s_l.[\mathtt{z}^{0;0}]^\circ$, for every k, l such that $k + l \geq 1$.
2. $[\mathtt{s}^{0;1}_0]^\circ \equiv \mathtt{Eb}^1[\mathtt{Ws0}]$.
3. $[\mathtt{s}^{0;1}_1]^\circ \equiv \mathtt{Eb}^1[\mathtt{Ws1}]$.
4. $[\mathtt{p}^{0;1}]^\circ \equiv \mathtt{Eb}^1[\mathtt{P}]$.
5. $[\pi^{k;l}_i]^\circ \equiv \backslash x_1 \ldots x_{k+l}.x_i$, with $1 \leq i \leq k + l$.
6. $[\mathtt{c}^{0,3}]^\circ \equiv \backslash xyz.\mathtt{B}\, x\, y\, z$.
7. Let $\vdash [f]^\circ : (\multimap^{k'}_{i=1} \$\mathbf{W}) \multimap (\multimap^{l'}_{i=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}$, and $\vdash [g_i]^\circ : (\multimap^{k}_{i=1} \$\mathbf{W}) \multimap \$^{m_i}\mathbf{W}$, with $i \in \{1, \ldots, k'\}$, and $\vdash [h_j]^\circ : (\multimap^{k}_{i=1} \$\mathbf{W}) \multimap (\multimap^{l_j}_{i=1} \$^{n_j}\mathbf{W}) \multimap \$^{n_j}\mathbf{W}$, with $j \in \{1, \ldots, l'\}$. If $p = \max\{m, m_1, \ldots, m_{k'}, n_1, \ldots, n_{l'}\}$, and $l = \max\{l_1, \ldots, l_{l'}, l'\}$, then:
$$\begin{aligned}
&[\circ^{k;l}_{k';l'}[f, g_1, \ldots, g_{k'}, h_1, \ldots, h_{l'}]]^\circ \equiv \\
&\quad \curlyvee^{k;l\backslash l}_{2p+1}\, [\boxdot^{k;l}_{k'}[\backslash x_1 \ldots x_{k'}.\mathtt{Ee}^{p-m}_{k';l'}[[f]^\circ](\mathtt{El}^1_1[\mathtt{Coerce}^{p-m-1}]\, x_1) \ldots (\mathtt{El}^1_1[\mathtt{Coerce}^{p-m-1}]\, x_{k'}) \\
&\qquad\qquad , \backslash x_1 \ldots x_k.\mathtt{Ee}^{p-m_1}_{k;0}[[g_1]^\circ](\mathtt{El}^1_1[\mathtt{Coerce}^{p-m_1-1}]\, x_1) \ldots (\mathtt{El}^1_1[\mathtt{Coerce}^{p-m_1-1}]\, x_k) \\
&\qquad\qquad \ldots \\
&\qquad\qquad , \backslash x_1 \ldots x_k.\mathtt{Ee}^{p-m_{k'}}_{k;0}[[g_{k'}]^\circ](\mathtt{El}^1_1[\mathtt{Coerce}^{p-m_{k'}-1}]\, x_1) \ldots (\mathtt{El}^1_1[\mathtt{Coerce}^{p-m_{k'}-1}]\, x_k) \\
&\qquad\qquad , \backslash x_1 \ldots x_k.\mathtt{Ee}^{p-n_1}_{k;l_1}[[h_1]^\circ](\mathtt{El}^1_1[\mathtt{Coerce}^{p-n_1-1}]\, x_1) \ldots (\mathtt{El}^1_1[\mathtt{Coerce}^{p-n_1-1}]\, x_k) \\
&\qquad\qquad \ldots \\
&\qquad\qquad , \backslash x_1 \ldots x_k.\mathtt{Ee}^{p-n_{l'}}_{k;l_{l'}}[[h_{l'}]^\circ](\mathtt{El}^1_1[\mathtt{Coerce}^{p-n_{l'}-1}]\, x_1) \ldots (\mathtt{El}^1_1[\mathtt{Coerce}^{p-n_{l'}-1}]\, x_k)]] \ .
\end{aligned}$$
8. If $\vdash [f_i]^\circ : \$\mathbf{W} \multimap (\multimap^{k}_{i=1} \$\mathbf{W}) \multimap (\multimap^{l}_{i=1} \$^{m_i}\mathbf{W}) \multimap \$^{m_i}\mathbf{W} \multimap \$^{m_i}\mathbf{W}$, with $i \in \{0, 1\}$, and $\vdash [g]^\circ : (\multimap^{k}_{i=1} \$\mathbf{W}) \multimap (\multimap^{l}_{i=1} \$^{m}\mathbf{W}) \multimap \$^{m}\mathbf{W}$, then:
$$[\mathtt{r}^{k+1;l}[g, f_0, f_1]]^\circ \equiv \mathtt{It}_{1+\mathsf{k};l}[F_0, F_1, G]\ ,$$
where $G \equiv \mathtt{Ee}^{p-m}_{k+1;l+1}[\backslash n_0\, n_1 \ldots n_k\, s_1 \ldots s_l\, r.[g]^\circ\, n_1 \ldots n_k\, s_1 \ldots s_l]$, $F_i \equiv \mathtt{Ee}^{p-m_i}_{k+1;l+1}[[f_i]^\circ]$, with $p = \max\{m_0, m_1, m\}$, and $i \in \{0, 1\}$.

Interpreting SRN to WALT. Let $\mathcal{R}$ be the set of environments, such that, every $\rho \in \mathcal{R}$ is a map from V to $\mathbb{N}$. Then, $[\![\ ]\!]$ is a map from a pair in $(\mathsf{SRN} \cup \Sigma_{\mathsf{SRN}}) \times \mathcal{R}$, to WALT, inductively defined on its first argument:
$$\begin{aligned}
[\![x]\!]_\rho &= [\![\rho(x)]\!]_\rho && (x \in \mathsf{V}) \\
[\![0]\!]_\rho &= [0]^\circ && \\
[\![f]\!]_\rho &= [f]^\circ && (f \in \Sigma_{\mathsf{SRN}}) \\
[\![f(t_1, \ldots, t_k, u_1, \ldots, u_l)]\!]_\rho &= && \\
&\mathtt{Ee}^{v-u+1-m}_{0;l}[\mathtt{Ee}^{u-1}_{0;k+l}[[\![f]\!]_\rho](\mathtt{El}^{u-p_1}_0[[\![t_1]\!]_\rho]) \ldots (\mathtt{El}^{u-p_k}_0[[\![t_k]\!]_\rho])] && \\
&\quad (\mathtt{El}^{v-q_1}_0[[\![u_1]\!]_\rho]) \ldots (\mathtt{El}^{v-q_l}_0[[\![u_l]\!]_\rho]) && (f \in \Sigma^{k,l}_{\mathsf{SRN}})
\end{aligned}$$

when $u = \max\{m, p_1, \ldots, p_k\}$, $v = \max\{u - 1 + m, q_1 \ldots, q_l\}$, and:

$$\begin{array}{ll} \vdash [\![f]\!]_\rho : (\multimap\!\!\bullet_{i=1}^{k} \$\mathbf{W}) \multimap\!\!\bullet (\multimap\!\!\bullet_{j=1}^{l} \$^m\mathbf{W}) \multimap\!\!\bullet \$^m\mathbf{W} & \\ \vdash [\![t_i]\!]_\rho : \$^{p_i}\mathbf{W} & i \in \{1, \ldots, k\} \\ \vdash [\![u_j]\!]_\rho : \$^{q_j}\mathbf{W} & j \in \{1, \ldots, l\} \ . \end{array}$$

Otherwise, $[\![\ ]\!]$ is undefined.

Weight of a term in SRN. For proving the statement that formalizes how we can embed SRN into WALT (Theorem 6.1 below) we need a notion of weight of a *closed* term in SRN, which, essentially, gives a measure of its impredicativity. For every *closed* term $t \in \mathsf{SRN} \cup \Sigma_{\mathsf{SRN}}$, $\mathrm{wg}(t)$ is the *weight of* t, defined by induction on t. If t is one among zero, predecessor, successor, projection, and branching, then $\mathrm{wg}(t) = 0$. Otherwise:

$$\begin{aligned} \mathrm{wg}(\circ^{k;l}_{k';l'}[f, g_1, \ldots, g_{k'}, h_1, \ldots, h_{l'}]) &= 3\max\{\mathrm{wg}(f), \mathrm{wg}(g_1), \ldots, \mathrm{wg}(g_k), \mathrm{wg}(h_1), \ldots, \mathrm{wg}(h_l), \frac{1}{3}\} \\ \mathrm{wg}(\mathbf{r}^{k+1;l}[g, h_0, h_1]) &= 2\max\{\mathrm{wg}(g), \mathrm{wg}(h_0), \mathrm{wg}(h_1), \frac{1}{2}\} \\ \mathrm{wg}(f(t_1, \ldots, t_k, u_1, \ldots, u_l)) &= 2\max\{\mathrm{wg}(f), \mathrm{wg}(t_1), \ldots, \mathrm{wg}(t_k), \mathrm{wg}(u_1), \ldots, \mathrm{wg}(u_l), \frac{1}{2}\} \end{aligned}$$

Theorem 6.1 (SRN is a subsystem of WALT.) *Let $k, l \in \mathbb{N}$, $f \in \Sigma^{k,l}_{\mathsf{SRN}}$, and $t, t_1, \ldots, t_k, u_1, \ldots, u_l$ be terms of SRN.*

1. *There is an $m \geq 1$ such that $\vdash [f]^\circ : (\multimap\!\!\bullet_{i=1}^{k} \$\mathbf{W}) \multimap\!\!\bullet (\multimap\!\!\bullet_{j=1}^{l} \$^m\mathbf{W}) \multimap\!\!\bullet \$^m\mathbf{W}$.*
2. *$[\![f(t_1, \ldots, t_k, u_1, \ldots, u_l)]\!]_\rho$ is defined, for every ρ.*
3. *$\vdash [\![t]\!] : \$^m\mathbf{W}$ with $m \leq \mathrm{wg}(t)$.*
4. *$[\![n]\!] \rightsquigarrow^+_w \overline{\overline{n}}$, for every $n \geq 0$.*
5. *If $f(n_1, \ldots, n_k, s_1, \ldots, s_l) = n$, then $[\![f(n_1, \ldots, n_k, s_1, \ldots, s_l)]\!] \rightsquigarrow^*_w \overline{\overline{n}}$, for every $n_1, \ldots, n_k, s_1, \ldots, s_l, n \in \mathbb{N}$.*

Point 1 is a direct consequence of the typing of the combinators of WALT that we use in the definition of $[f]^\circ$. Point 2 follows from point 1 here above and from the definition of $[\![\ ]\!]$. Point 3 holds by induction on t. Point 4 holds by induction on n. Point 5 holds by induction on f. Finally, by structural induction on t, we have:

Corollary 6.2 (The embedding of SRN into WALT is sound.) *Let $t \in \mathsf{SRN}$, and $n \in \mathbb{N}$. If $t = n$, then $[\![t]\!]_\rho \rightsquigarrow^+_w \overline{\overline{n}}$, for every environment ρ.*

7 Conclusions and future work

WALT is the first higher-order deductive system, derived from Linear logic, such that: (i) is sound and complete w.r.t. FP, (ii) is complete w.r.t. SRN, (iii) makes evident the layered nature of the almost flat normal/safe hierarchy about the arguments of the terms of SRN, and (iv) no constant symbol is required to obtain the point (iii), since every datatype can defined from scratch.

In particular, point (ii) allows to say that the less an argument of a term of SRN is "polynomially impredicative", the deeper its representation is inside the stratified structure of the derivations of WALT. This relation between the polynomial impredicativity and the stratification suggests that a relation between WALT and Higher type ramified recurrence (HTRR) [BNS00, BS01], or Higher linear ramified recursion (HOLRR) [DLMR04] should exist. We think that the most intriguing is the one between HTRR and WALT. The reason is that HTRR characterizes FP by a careful interplay of conditions about its types, built on an almost linear arrow type and !-modal types, and its terms, derived from Gödel System T [G5̈8]. The notions of complete/incomplete types, linked to their modality, the possibility of duplicating at will only ground types, and the affinability, which expresses linearity constraints on the bound variables of incomplete types, strongly recall the properties we enforce on $\multimapdotinv$, on its arguments and on the \$-modal assumptions of !-boxes in WALT.

A further investigation could go in the "backward" direction, namely from the structural proof-theoretical world, represented by WALT, to the recursive theoretical one, represented by SRN.

Let us look at Figure 11. It fixes a hierarchy, based on syntactic restrictions. We already know what

$$\begin{tikzcd}
& \mathsf{SRN} & \\
\mathsf{RlSRN} \arrow[ur, hook] & & \mathsf{ClSRN} \arrow[ul, hook] \\
& \mathsf{BC}^- \arrow[ul, hook] \arrow[ur, hook] &
\end{tikzcd}$$

Figure 11: A simple syntactic hierarchy

SRN, ClSRN, and BC$^-$ are. Instead, RlSRN, called *Recursion-linear* SRN, is "new". It is defined as the "complement" of ClSRN w.r.t. SRN by *restricting the recursive scheme* of SRN to one that uses its safe variables linearly, while leaving the composition scheme untouched.

SRN and ClSRN should be both polytime complete, as consequence of the moral equivalence "full composition scheme of SRN $\simeq$ recursion scheme + linear composition scheme of ClSRN", we have proved in this work.

Moreover, we know that BC$^-$ is contained into the class of deterministic logarithmic space [Nee04]. We can ask which is the space complexity of ClSRN, which should not coincide to the one of SRN, because they develop different computation processes of, very likely, FP. Of course, the same questions may be asked and answered about RlSRN, so inducing a space hierarchy, which originates from a syntactic analysis of SRN, in its turn coming from the structural proof-theoretic roots of WALT.

References

[AR02] A. Asperti and L. Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):137–175, 2002.

[Asp98] A. Asperti. Light affine logic. In *Proceedings of the 13th IEEE Symposium on Logic in Computer Science (LICS)*, pages 300–308, 1998.

[BC92] S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[BNS00] S. Bellantoni, K.-H. Niggl, and H. Schwichtenberg. Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic*, 104:17–30, 2000.

[BS01] S. Bellantoni and H. Schwichtenberg. Feasible computation with higher types. Marktoberdorf Summer School Proceedings, 2001.

[BW96] A. Beckmann and A. Weiermann. A term rewriting characterization of the polytime functions and related complexity classes. *Archive for Mathematical Logic*, 36(1):11 – 30, December 1996.

[Cob65] A. Cobham. The intrinsic computational difficulty of functions. In *Proceedings of 1964 International Congress for Logic, Methodology and Philosophy of Sciences*, pages 24–30, 1965.

[Der82] D. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.

[DLMR04] U. Dal Lago, S. Martini, and L. Roversi. Higher-order linear ramified recurrence. In *Proceedings of TYPES'04*, volume 3085 of *Lecture Notes in Computer Science*, pages 178 – 193. Springer Verlag, December 2004.

[G5̈8] K. Gödel. Über eine bisher noch nicht benützte erweiterung des finiten standpunktes. *Dialectica*, 12:280–287, 1958.

[Gir98] J.-Y. Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.

[Hof97] M. Hofmann. A mixed modal/linear lambda calculus with applications to bellantoni-cook safe recursion. In *Proceedings of the 11th International Workshop on Computer Science Logic*, pages 275–294, 1997.

[Hof99a] M. Hofmann. Linear types and non-size-increasing polynomial time computation. In *Logic in Computer Science*, pages 464–473, 1999.

[Hof99b] M. Hofmann. *Type systems for polynomial-time computation*. Habilitationsschrift, Darmstadt University of Technology, 1999.

[Hof00] M. Hofmann. Safe recursion with higher types and BCK-algebra. *Annals of Pure and Applied Logic*, 104:113–166, 2000.

[Hue80] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[Jon99] N. D. Jones. Logspace and ptime characterized by programming languages. *Theoretical Computer Science*, 228:151–174, 1999.

[Laf04] L. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318:163–180, 2004. Special issue on Implicit Computational Complexity.

[Lei93] D. Leivant. Stratified functional programs and computational complexity. In *Proceedings of 20th ACM Symposium on Principles of Programming Languages*, pages 325–333, 1993.

[Lei94] D. Leivant. A foundational delineation of poly-time. *Information and Computation*, 110(2):391–420, 1994.

[Lei95] D. Leivant. Ramified recurrence and computational complexity I: word recurrence and poly-time. In *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1995.

[Lei99] D. Leivant. Ramified recurrence and computational complexity III: Higher type recurrence and elementary complexity. *Annals of Pure and Applied Logic*, 96:209–229, 1999.

[LM] D. Leivant and J.-Y. Marion. Predicative recurrence and computational complexity IV: Predicative functionals and poly-space. should be published !

[LM94] D. Leivant and J.-Y. Marion. Ramified recurrence and computational complexity II: Substitution and poly-space. In *Proceedings of 8th International Workshop on Computer Science Logic (CSL)*, pages 486–500, 1994.

[MO04] A. S. Murawski and C.-H. L. Ong. On an interpretation of safe recursion in light affine logic. *Theor. Comput. Sci.*, 318(1-2):197–223, 2004.

[Nee04] P. M. Neergaard. A functional language for logarithmic space. In *Prog. Lang. and Systems: 2nd Asian Symp. (APLAS 2004)*, volume 3302 of *LNCS*, pages 311–326. 2004, Springer-Verlag, November 2004.

[Rov07] L. Roversi. Weak Affine Light Typing: Intensional expressivity, Polytime soundness and completeness. Technical Report 103/07, Dipartimento di Informatica, Torino, C.so Svizzera, n.185 — 10149 Torino — Italy, December 2007.

[Ter01] K. Terui. Light affine lambda calculus and polytime strong normalization. In *Proceedings of the 16th Annual IEEE Conference on Logic in Computer Science (LICS01)*, pages 209–220, 2001.

[Ter07] K. Terui. Light affine lambda calculus and polynomial time strong normalization. *Archive for Mathematical Logic*, 46(3–4):253–280, 2007.

A Some detailed proofs

A.1 $\curlyvee^{\mathsf{n};\mathsf{p}\backslash i}_{m}[M]$ is well typed.

For every p, this can be proved by cases on the value of p, and by induction on i. When $\mathsf{p} \leq 1$ or $i = 0$, $\curlyvee^{\mathsf{n};\mathsf{p}\backslash i}_{m}[M]$ is M with, at most, a single safe argument. So, the statement trivially holds. Let us assume $\mathsf{p} > 1$ and $\mathsf{p} \geq i \geq 1$.

The base case has $i = 1$. We start from the type $(\multimap^{\mathsf{n}}_{i=1} \$\mathbf{W}) \multimap (\multimap^{\mathsf{p}^2}_{j=1} \$^m\mathbf{W}) \multimap \$^m\mathbf{W}$ of M, observing that:

$$\mathsf{p}^2 = \sum_{k=1}^{\mathsf{p}} \mathsf{p} = \mathsf{p} + \sum_{k=1}^{\mathsf{p}-1} \mathsf{p} = \mathsf{p} - 1 + 1 + \sum_{k=1}^{\mathsf{p}-1} \mathsf{p} = (1 + \sum_{k=1}^{\mathsf{p}-1} \mathsf{p}) + (\mathsf{p} - 1)$$

So, we can apply the clause defining $\curlyvee^{\mathsf{n};\mathsf{p}\backslash 1}_{m}[M]$, getting that its type is the one of

$$\circlearrowleft\mathtt{mY}^{\mathsf{n};(1+\sum_{k=1}^{\mathsf{p}-1}\mathsf{p},\mathsf{p}-1)}_{m}[M] \ ,$$

namely $(\multimap^{\mathsf{n}}_{j=1} \$\mathbf{W}) \multimap (\multimap^{1+\sum_{k=1}^{\mathsf{p}-1}\mathsf{p}}_{j=1} \$^{m+4(\mathsf{p}-1)}\mathbf{W}) \multimap \$^{m+4(\mathsf{p}-1)}\mathbf{W}$.

By induction, the type of $\curlyvee_m^{n;p\backslash i-1}[M]$ is $(\multimap_{j=1}^{n} \$\mathbf{W}) \multimap (\multimap_{j=1}^{(i-1)+\sum_{k=1}^{p-(i-1)} p} \$^{m+4(p-1)(i-1)}\mathbf{W}) \multimap \$^{m+4(p-1)(i-1)}\mathbf{W}$. Observing that the following of equivalences hold:

$$(i-1)+\sum_{k=1}^{p-(i-1)} p = (i-1)+\sum_{k=1}^{p-i+1} p = (i-1)+p+\sum_{k=1}^{p-i} p = i-1+p-1+1+\sum_{k=1}^{p-i} p = (i+\sum_{k=1}^{p-i} p)+(p-1)$$

we can transform the type of $\curlyvee_m^{n;p\backslash i-1}[M]$ so that we can use it as argument of $\circlearrowleft\mathtt{mY}_{m+4(p-1)(i-1)}^{n;(i+\sum_{k=1}^{p-i},p-1)}$. By definition, we get a term with the type we need.

A.2 $\curlyvee_m^{n;p\backslash i}[M]$ well behaves.

Let $p = 0$. Then,

$$\curlyvee_m^{n;0\backslash i}[M]\,\overline{\overline{n_1}}\ldots\overline{\overline{n_n}}\,\overbrace{\overline{\overline{s_{0-i+1}}}\ldots\overline{\overline{s_0}}}^{i}\,\overbrace{\overline{\overline{s_1}}\ldots\overline{\overline{s_1}}}^{0}\ldots\ldots\overbrace{\overline{\overline{s_i}}\ldots\overline{\overline{s_i}}}^{0} \equiv M\,\overline{\overline{n_1}}\ldots\overline{\overline{n_n}} \tag{5}$$

where the sequences of safe arguments cannot exist since we assume that the indices of the safe arguments start from 1. So, (5) rewrites to $M\,\overline{\overline{n_1}}\ldots\overline{\overline{n_n}}$ in 0 steps.

Let $p \geq 1$ and $i = 0$. Then,

$$\curlyvee_m^{n;p\backslash 0}[M]\,\overline{\overline{n_1}}\ldots\overline{\overline{n_n}}\,\overbrace{\overline{\overline{s_{p-0+1}}}\ldots\overline{\overline{s_p}}}^{0}\,\overbrace{\overline{\overline{s_1}}\ldots\overline{\overline{s_1}}}^{p}\ldots\ldots\overbrace{\overline{\overline{s_{p-0}}}\ldots\overline{\overline{s_{p-0}}}}^{p} \equiv M\,\overline{\overline{n_1}}\ldots\overline{\overline{n_n}}\,\overbrace{\overline{\overline{s_1}}\ldots\overline{\overline{s_1}}}^{p}\ldots\ldots\overbrace{\overline{\overline{s_p}}\ldots\overline{\overline{s_p}}}^{p} \tag{6}$$

where the inital sequences of safe arguments are those required directly by M. This is why $\curlyvee_m^{n;p\backslash 0}[M]$ coincides to M and the statement holds relatively (6).

Let $p \geq 1$ and $i > 0$. By induction, we have:

$$\curlyvee_m^{n;p\backslash i-1}[M]\,\overline{\overline{n_1}}\ldots\overline{\overline{n_n}}\,\overbrace{\overline{\overline{s_{p-(i-1)+1}}}\ldots\overline{\overline{s_p}}}^{i-1}\,\overbrace{\overline{\overline{s_1}}\ldots\overline{\overline{s_1}}}^{p}\ldots\ldots\overbrace{\overline{\overline{s_{i-1}}}\ldots\overline{\overline{s_{i-1}}}}^{p} \rightsquigarrow_w^* M\,\overline{\overline{n_1}}\ldots\overline{\overline{n_n}}\,\overbrace{\overline{\overline{s_1}}\ldots\overline{\overline{s_1}}}^{p}\ldots\ldots\overbrace{\overline{\overline{s_p}}\ldots\overline{\overline{s_p}}}^{p}$$

We also know that, for every term N with the right type, depending on m':

$$\begin{aligned}&\circlearrowleft\mathtt{mY}_{m'}^{n;(i-1+\sum_{k=1}^{p-(i-1)} p,p)}[N]\,\overline{\overline{n_1}}\ldots\overline{\overline{n_n}}\,\overbrace{\overline{\overline{s_{p-(i-1)}}}\,\overline{\overline{s_{p-(i-1)+1}}}\ldots\overline{\overline{s_p}}}^{i-1}\,\overbrace{\overline{\overline{s_1}}\ldots\overline{\overline{s_1}}}^{p}\ldots\ldots\overbrace{\overline{\overline{s_{p-(i-1)-1}}}\ldots\overline{\overline{s_{p-(i-1)-1}}}}^{p}\\ &\rightsquigarrow_w^* N\,\overline{\overline{n_1}}\ldots\overline{\overline{n_n}}\,\overbrace{\overline{\overline{s_{p-(i-1)+1}}}\ldots\overline{\overline{s_p}}}^{i-1}\,\overbrace{\overline{\overline{s_1}}\ldots\overline{\overline{s_1}}}^{p}\ldots\ldots\overbrace{\overline{\overline{s_{p-(i-1)-1}}}\ldots\overline{\overline{s_{p-(i-1)-1}}}}^{p}\overbrace{\overline{\overline{s_{p-(i-1)}}}\ldots\overline{\overline{s_{p-(i-1)}}}}^{p}\end{aligned} \tag{7}$$

So, in (7), N can be $\curlyvee_m^{n;p\backslash i-1}[M]$ with $m' = m + 4(p-1)(i-1)$. But, by definition,

$$\circlearrowleft\mathtt{mY}_{m+4(p-1)(i-1)}^{n;(i-1+\sum_{k=1}^{p-(i-1)} p,p)}[\curlyvee_m^{n;p\backslash i-1}[M]]$$

is $\circlearrowleft\mathtt{mY}_m^{n;(i+\sum_{k=1}^{p-i} p,p)}[M]$, hence with the behaviour we want.